Cryptographic Hash Functions: Recent Design Trends and Security Notions*

Saif Al-Kuwari[1] James H. Davenport[2] Russell J. Bradford[3]

Department of Computer Science,
University of Bath, Bath, BA2 7AY, UK
[1]S.Alkuwari@bath.ac.uk, [2]J.H.Davenport@bath.ac.uk,
[3]R.J.Bradford@bath.ac.uk

Abstract

Recent years have witnessed an exceptional research interest in cryptographic hash functions, especially after the popular attacks against MD5 and SHA-1 in 2005. In 2007, the U.S. National Institute of Standards and Technology (NIST) has also significantly boosted this interest by announcing a public competition to select the next hash function standard, to be named SHA-3. Not surprisingly, the hash function literature has since been rapidly growing in an extremely fast pace. In this paper, we provide a comprehensive, up-to-date discussion of the current state of the art of cryptographic hash functions security and design. We first discuss the various hash functions security properties and notions, then proceed to give an overview of how (and why) hash functions evolved over the years giving raise to the current diverse hash functions design approaches.

*A short version of this paper is in [1]. This version has been thoroughly extended. An identical version has been uploaded to the Cryptology ePrint Archive: eprint.iacr.org/2011/565

Contents

1 Introduction **3**

I Hash Functions Security **4**

2 Security Properties **4**
2.1 Collision-Resistance (CR) . . . 5
2.2 Pre-image Resistance (Pre) . . . 5
2.3 2nd Pre-image Resistance (Sec) . . . 6
2.4 Other Properties . . . 6

3 Security Notions **7**
3.1 Indifferentiability from $\mathcal{RO}$ . . . 7
3.2 Indistinguishability from PRF . . . 11
3.3 Unforgeability . . . 11
3.4 Other Notions . . . 12

4 Multi-Property-Preserving **12**

5 Cryptographic Proofs **14**

II Hash Functions Design **14**

6 Keyless vs. Keyed Hash Functions **15**

7 Iterative Hash Functions **15**
7.1 Merkle-Damgård Construction . . . 15
7.2 Generic Attacks Against Merkle-Damgård . . . 16
7.3 Variants of Merkle-Damgård . . . 19
7.4 Sponge Construction . . . 24

8 Tree-based Hash Functions **25**

9 Compression Functions **25**
9.1 Hash Functions Based on Block and Stream Ciphers . . . 26
9.2 Hash Functions Based on Mathematical Problems . . . 27
9.3 Other Approaches . . . 27

10 Conclusion and Summary **28**

1 Introduction

Cryptographic hash functions have indeed proved to be the workhorses of modern cryptography. Their importance was first realised with the invention of public key cryptography (PKC) by Diffie and Hellman [43] in 1976, where it became an integral part of PKC ever since. Unfortunately, recent advances in cryptanalysis revealed inherent weaknesses in most of the popular hash functions triggering an urgent call for further research in this area. In response, two main approaches have been adopted: either *patching* the existing constructions by slightly modifying them to fix a particular set of weaknesses, or designing new hash functions from scratch. In the first approach, if inherent weaknesses were discovered, they imply that the design principles on which the hash function is based are flawed and unless they are thoroughly revised, it is most likely that those weaknesses will still exist, even if they appear to have been fixed by some minor modifications. Likewise, in the second approach, if a hash function is designed from scratch, it may sufficiently resist a particular set of weaknesses, but may also covertly suffer from other (possibly more severe) weaknesses that might not have been spotted at early development stages. Existing constructions, on the other hand, have the advantage that they have been extensively studied and analysed over time, thus, unless very carefully designed, structurally new hash functions may well be susceptible to more attacks than those that they resist.

Hash functions are essentially many-to-one functions since they map arbitrary length inputs to fixed length outputs and the input is usually larger than the output (hash functions are compressing primitives). Thus, collisions (different messages hashing to the same value) in hash functions are unavoidable due to the pigeonhole principle[1]. Yuval [117] was the first to discuss how to find collisions in hash functions using the Birthday Paradox, leading to what is commonly known today as the *birthday attack*. In this attack, a collision is found with probability $q^2/2^n$ after q queries to a hash function outputting values of length n-bit [18]. While collision resistance is certainly an important property that hash functions are expected to possess, it is not the only one, and in some applications it is not even required. Pre-image resistance (non-invertibility), for example, is a more difficult and more practical property. In fact, in most applications it is more devastating to be able to invert a hash value, than finding a collision between two arbitrary messages. Therefore, it is the application at which the hash function is being used that determines the security properties that it should preserve. In this paper, we present and discuss hash functions' most common security properties and notions followed by a thorough discussion of the different approaches that were adopted to build hash functions, which, of course, the security of hash functions have influenced significantly as we will show.

SHA-3 Competition. For a long time, SHA-1 and MD5 hash functions have been the closest to a hashing de facto, this, however, has changed in 2004 and 2005 when Wang *et al.* [109, 110, 111] showed that finding collisions for MD5 can be easy while substantially reducing the work needed to find collisions on SHA-1 to 2^{69}, which is much less than the expected 2^{80}. Although a break with complexity of 2^{69} is still (at the time of writing) theoretical, it showed that SHA-1 is not as strong and collision-resistant as it is supposed (and thought) to be. This has driven the National Institute of Standards and Technology (NIST) in November 2007 to announce an open competition[2] to select a new hash functions standard, to be named SHA-3 [80]. NIST received 64 submissions, 51 of which were accepted for round 1 of the competition in December 2008. In July 2009, only 14 round 1 candidates successfully progressed to round 2; these are briefly analysed in [5]. In December 2010, the 5 finalist candidates where chosen (these are,

[1]The pigeonhole principle states that if m pigeons are distributed over n holes, and $m > n$, then there is at least one hole with more than one pigeon.

[2]For comprehensive resource about SHA-3 competition and all its candidates, see http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo

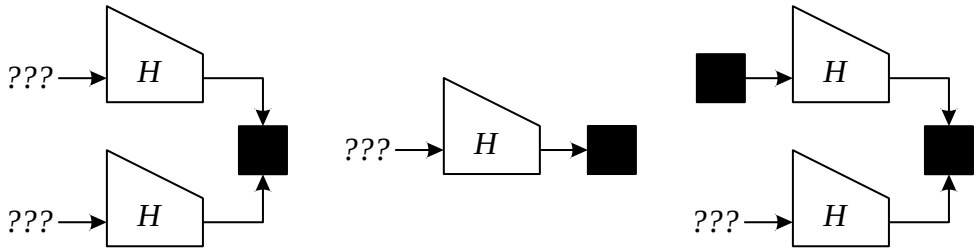
BLAKE, Grøstl, JH, Keccak and Skein), and the winner is expected to be announced in the second quarter of 2012

Paper Organisation. This paper consists of two main parts. In this first part, we discuss security of hash functions as follows. First, we provide a discussion about the three classical hash function security properties in section 2, namely collision resistance in section 2.1, pre-image resistance in section 2.2 and 2nd pre-image resistance in section 2.3, as well as some other properties (including statistical and application-specific ones) in section 2.4. We then describe in length the indifferentiability framework in section 3.1. Brief descriptions of the indistinguishability from Pseudorandom Function (PRF) and the unforgeability notions are provided in sections 3.2 and 3.3, respectively. We also talk about some other security notions in 3.4. We conclude the first part In section 4, where we discuss the multi-property-preserving paradigm (MPP) and give a few examples of recent MPP constructions. In the second part, we discuss the design of hash functions as follows. In section 6, we classify hash functions as keyed and keyless. Section 7 then provides a relatively lengthy and up-to-date discussion about various iterative hash functions, this is indeed the most common approach (at least contemporarily) in designing hash functions. In particular, we discuss the Merkle-Damgård construction in section 7.1, generic attacks against Merkle-Damgård in section 7.2, and how accordingly the research community tried to patch the construction in section 7.3. Beside iterative functions, tree-based ones have also been proposed, these are briefly discussed in section 8. We then discuss compression function design in section 9, in particular hash functions based on block/stream ciphers in section 9.1, and provably secure (based on mathematical problems) hash functions in section 9.2.

Part I
Hash Functions Security

2 Security Properties

The basic (classical) properties a hash function is expected to preserve are: collision resistance, pre-image resistance and 2nd pre-image resistance; figure 1 illustrates these properties graphically. Although these are thought to be the universal security properties that most hash functions should preserve, there may be other application-specific security properties that hash functions should additionally (or instead) preserve if they are to be used in a given application. In this section we provide an elaborate discussion on the basic definitions of these properties. Generally, when we say that an attack succeeds in breaking a particular hash function, that does not necessarily mean that the hash function is deemed broken in practice. If an attack succeeds to prove that a hash function can be exploited (e.g. finding a collision, or pre-image, or second pre-image) with work less than that required by the birthday or brute force attack, the hash function is considered broken, even if the work required to break it is still infeasible in practice (this is called a theoretical break). Indeed, finding such flaws in a hash function is an evidence of structural weaknesses that may be exploited at later stages to turn this theoretical break into a practical one; the primary example is MD5, which was first theoretically broken, then the attacks eventually evolved and today practical collisions can easily be found in MD5 [64, 103].



(a) Collision resistance (b) Pre-image resistance (c) 2nd pre-image resistance

Figure 1: Graphical representation of the collision resistance, pre-image resistance and 2nd pre-image resistance properties

2.1 Collision-Resistance (CR)

A hash collision occurs when two (arbitrary) different messages hash to the same value. That is, for a collision resistant hash function H, it should be computationally infeasible to find any two messages M and M' such that $H(M) = H(M')$ while $M \neq M'$. This also applies to families of hash function (i.e. keyed hash functions, where members of the family are indexed by different keys). Formally, the advantage of an adversary A of finding a collision in a hash function H is defined as follows:

$$\mathbf{Adv}^{\mathrm{cr}}_{H}(A) = \Pr\left[(M, M') \xleftarrow{\$} A : M \neq M' \wedge H(M) = H(M')\right]$$

For a secure hash function, the best attack to find a collision should not be better than the birthday attack (i.e. not better than work complexity of $2^{n/2}$ for a hash function outputting n-bit hash values). Collision resistance was first formally defined by Damgård [37], and is sometimes called *Strong Collision Resistance*. Some authors use the term *multi-block collision* to refer to 2 colliding messages, each consisting of at least 2 blocks. This is not to be confused with multi-collision, where multiple messages collide at the same hash value regardless of their sizes (sometimes also called K-collision, where K is the number of the colliding messages). Finding a K-collision should have a complexity of at least $2^{(K-1)n/K}$. In [93], Rogaway discussed the *foundation-of-hashing dilemma* which states that collision resistance cannot be formally defined for keyless hash functions, that is, there will always be collisions, it is just that us, humans, may not be able to find them, but such a concept cannot be formalised mathematically for keyless hash functions. Rogaway proposed a solution to this dilemma by means of explicit security reduction, which he called the *human-ignorance* framework.

2.2 Pre-image Resistance (Pre)

For all practical purposes, hash functions should be computationally *non-invertible*. When a message is hashed, it should be (computationally) infeasible to retrieve the original message from which the hash value was obtained. That is, for a pre-image resistant hash function H, given a hash value $H(M)$ of a particular message M, it should be computationally infeasible to retrieve the original message M, or indeed generate any message $M' \neq M$ such that $H(M') = H(M)$. Succinctly,

$$\mathbf{Adv}^{\mathrm{pre}[m]}_{H}(A) = \Pr\left[M \xleftarrow{\$} \{0,1\}^m; Y \leftarrow H(M); M' \xleftarrow{\$} A(Y) : H(M') = Y\right]$$

For a hash function to be pre-image resistant, the best attack against the hash function should be the brute force attack (i.e. work complexity of 2^n operations for a hash function with output size

n). Pre-image resistance is also sometimes called *One-wayness*. Generally, collision resistance *does not* guarantee pre-image resistance [37], but in [94] it was shown that pre-image resistance can be implied by collision resistance if the hash function was sufficiently compressing (i.e. its domain is much larger than its range). Similarly, Stinson [105] argued that it is possible to obtain good reduction from collision resistance to pre-image resistance under several assumptions, but he then showed that these assumptions are difficult/impossible to satisfy in practice. Stinson also introduced the zero pre-image notion, where an attacker finds the message M such that $H(M) = y = 0$. He remarked that there is no obvious reduction between zero-pre-image (where a specific value of y is inverted) and normal pre-image (where a random value of y is inverted), and that one may be harder or easier to find than the other.

2.3 2nd Pre-image Resistance (Sec)

Given a 2nd pre-image resistant hash function H, and a message M, it should be computationally infeasible for an adversary A to find a different message M' such that $M \neq M'$ and both M and M' hash to the same value, $H(M) = H(M')$. Succinctly,

$$\mathbf{Adv}_H^{\mathrm{sec}[m]}(A) = \Pr\left[M \xleftarrow{\$} \{0,1\}^m; M' \xleftarrow{\$} A(M) : M \neq M' \wedge H(M) = H(M')\right]$$

For H to be considered 2nd pre-image resistant, the best attack against H should be the brute force attack (i.e. with work complexity of 2^n for a hash function with output size n). 2nd pre-image resistance is also sometimes called *Weak Collision Resistance*. Although it is frequently claimed in the literature that collision resistance implies 2nd pre-image resistance [73, 94], Contini *et al.* [32] argued that interpreting some formal definitions of 2nd pre-image resistance in the literature (e.g. [73]) invalidates this claim, but generally this claim is true if both collision and 2nd pre-image resistance are defined properly. Like collision and K-collisions, a generalisation of 2nd pre-image is K-way 2nd pre-image where, given a message M and its hash value $H(M)$, an adversary A finds K different messages colliding in $H(M)$, that is, $H(M_1) = H(M_2) = \cdots = H(M_K) = H(M)$, while $M_1 \neq M_2 \neq \cdots \neq M_K \neq M$. Similarly, K-way pre-image occurs when, given $H(M)$, an adversary A finds K different messages colliding in $H(M)$. Finding a K-way (2nd) pre-image should have a complexity of at least $K.2^n$.

Remark. In [113], Yasuda showed that the compression function of a Merkle-Damgård hash function does not have to be CR for the whole hash function to be Sec or Pre. The author argued that it only suffices for a compression function to preserve weaker-than-CR properties, namely cs-SPR (chosen suffix second pre-image) and cs-OW (chosen suffix one-wayness), for the corresponding hash function to preserve Sec and Pre, respectively. This is indeed an important result since many compression functions used with the Merkle-Damgård construction were recently found to be not CR, which directly implies that their corresponding hash functions are not CR [74, 38], but that does not necessarily mean that they are also not Sec and/or Pre when their compression functions are modelled as cs-SPR and cs-OW (which are weaker than CR).

2.4 Other Properties

Other desirable properties that hash functions should preferably preserve include [73] (some of these properties are application specific, i.e., some hash function applications may not require some of these properties):

- Near-collision resistance: hash values of two different messages should differ significantly (even if the messages differ slightly), that is, a near-collision occurs if for two different messages $M \neq M'$, $H(M)$ differs from $H(M')$ by only a small number of bits.

- Semi-free-start collision resistance: a semi-free-start collision occurs when two different messages with the same (but random) IV hash to the same value. In practice, though, hash functions are usually specified with fixed IVs.
- Pseudo-collision resistance: a pseudo-collision (or free-start collision) occurs when it is possible to find a collision between two messages by only controlling their IVs (again, this attack is not practically relevant because most hash functions are shipped with fixed IVs, so an attacker cannot control the IV in practice). A variant of this property is pseudo-near-collision which results in a near-collision.
- Partial pre-image resistance: also sometimes called local one-wayness, states that it should be equally difficult to retrieve part of the original message from its hash value as retrieving the whole message, even if a portion of the message is already known.
- Non-correlation (correlation freeness): hash function inputs and outputs should not be statistically correlated; that is, even a small change in the input should drastically affect the output bits; this phenomenon is called the avalanche effect.
- Chosen Target Forced Prefix (CTFP) pre-image resistant [61]: applications that need to resist the herding attack [61] (see section 7.2) need to preserve this property which prevents an attacker from finding a string S such that given P and H, then $H(P||S) = F$ where F is a hash value computed before learning P.

A particularly problematic situation arises when trying to evaluate the security of sponge-based constructions [21] (see section 7.4). Traditionally, security bounds are based on the function's output length n, where collision requires $2^{n/2}$ and pre-image/2nd pre-image require 2^n. However, the sponge construction produces a variable length output. Realising this problem, the authors of the sponge construction introduced a reference security model, called the *Random Sponge*, which they use in their security analysis. Detailed discussion of the security of the sponge construction is beyond the scope of this paper.

3 Security Notions

Other than the basic (classical) security properties above, it has been suggested that hash functions should also preserve several other properties, most of which are usually application dependent. In this section, we try the shed the light on the most popular hash functions security notions that are rapidly becoming common requirements for most applications.

3.1 Indifferentiability from $\mathcal{RO}$

Security analysis and proofs of many cryptosystems are carried out in the random oracle model (ROM) [15], which assumes the presence of a *Random Oracle* ($\mathcal{RO}$). A $\mathcal{RO}$ is an abstract ideal primitive that returns infinite random response every time it is queried [15], though such response is usually truncated. Responses of $\mathcal{RO}$ are consistent for similar queries (a particular query will always receive the same $\mathcal{RO}$ response regardless of when and how many times it is made) and since $\mathcal{RO}$ is an atomic entity (i.e. it cannot be decomposed), it is often said to be *monolithic*. In practice, $\mathcal{RO}$s do not exist [27] and are instead instantiated by hash functions which are *not* monolithic by nature since hash functions are structured entities that usually process messages by repeatedly and iteratively calling an underlying primitive (commonly, a compression function).

Therefore, for proofs in the ROM[3] to hold in practice, the adopted hash function should *emulate* a $\mathcal{RO}$. A hash function H emulating a $\mathcal{RO}$ in this sense implies that H cannot be differentiated from a genuine $\mathcal{RO}$ and that systems proven secure in the ROM will remain secure if the $\mathcal{RO}$ is replaced by H. In the context of cryptography, there is a distinction between *indifferentiability* and *indistinguishability*. In indistinguishability, a distinguisher algorithm D is merely given black-box[4] access to the two systems. In indifferentiability, on the other hand, D is further given access the underlying primitives of the systems and can query them independently. Thus, indifferentiability is a generalisation of indistinguishability.

In [33] Coron *et al.*, based on Maurer's indifferentiability framework [71], introduced their popular hash function indifferentiability from $\mathcal{RO}$ framework, which can be used to prove that a hash function, with access to an ideal compression function, is indifferentiable from a genuine $\mathcal{RO}$. In this framework, the two building blocks of a hash function, namely a construction C and an ideal compression function $\mathcal{G}$, constitute a system (System 1) and the random oracle $\mathcal{F}$ (which the hash function needs to emulate) constitutes another system (System 2), then a distinguisher algorithm D with oracle access to both systems tries to challenge the systems and distinguish between them. If we do not introduce an extra component in System 2, D can easily distinguish between the two systems since System 1 consists of two components while System 2 consists of only one component. Thus, we introduce a simulator S in System 2 to simulate the ideal compression function $\mathcal{G}$ of System 1. The simulator S should be defined very carefully to simulate not only a conventional compression function operation such that given an input it returns a random output, but also how to behave consistently with the way C and $\mathcal{G}$ interact to handle D's queries. This is indeed a non-trivial task for S because all messages sent to C will be processed by $\mathcal{G}$ (i.e. $\mathcal{G}$ can see all messages sent to System 1, including those sent to C), but that does not hold for S in System 2 because the random oracle $\mathcal{F}$ is an atomic component and will return its responses independently from S (i.e. messages sent to $\mathcal{F}$ are not observable by S). It is important to note that D here challenges the systems by sending multiple queries to the different components of the systems and observes the responses, D then distinguishes between the systems based on their overall observed behaviour not just the individual responses of the queries. That is, if D sent a query to both systems and received different responses, that does not necessarily mean that D has succeeded in the distinguishing game because both responses are still random. However, if D observed a pattern in a series of responses from a particular system but did not observe similar behaviour in the other system, then it is apparent that one of them is behaving differently than the other and D succeeds in the distinguishing game. Although D does not necessarily have to tell which system is which, it will be obvious that the system that behaves in a more random manner is the $\mathcal{RO}$ system. Figure 2 illustrates the general setting where D has oracle access to both Systems 1 and 2.

Systems 1 and 2 are also sometimes called the *Real System* and the *Random System*, respectively. In System 1, C has oracle access to $\mathcal{G}$, where in System 2, S has oracle access to $\mathcal{F}$. However, while in System 1, C always queries $\mathcal{G}$ to obtain responses for any query it receives, in System 2, S may choose to query $\mathcal{F}$ or generate its responses uniformly at random. For Systems 1 and 2 to be indifferentiable from each other, it is important that S is programmed in such a way that $\mathcal{F}$ and S behave consistently with how C and $\mathcal{G}$ behave (note that S is the only customisable component). To illustrate this point, let's look at how $\mathcal{G}$ (which S should simulate) behaves. When $\mathcal{G}$ receives a query (which may be from C or D) it merely generates a random response since it is modelled as an ideal primitive. Having to simulate $\mathcal{G}$, the simulator

[3]Even though proofs in the ROM may not *always* guarantee security when the $\mathcal{RO}$ is instantiated by a practical hash function, if a scheme is proven secure in the ROM, there is strong evidence that this scheme exhibits sound structure, or at least do not suffer from serious inherent structural weaknesses.

[4]When an adversary has a black-box access to a component G it can only send queries to that component and receive responses back cannot access the internal components of G.

S, in turn, should do the same, but S is not an ideal primitive, so it can either query $\mathcal{F}$ to get a random oracle response (i.e. a response from an ideal primitive), or just return a uniformly random response from some randomness generation source. Since it is usually more expensive to query $\mathcal{F}$, S will always return uniformly random responses, unless it really has to query $\mathcal{F}$; when exactly to query or not to query $\mathcal{F}$ is (the main) part of the simulator's definition and depends on how System 1 behaves. In the case of a Merkle-Damgård hash function (when C is modelled as a Merkle-Damgård construction), the main differences between System 1 and System 2 can be summarised as follows:

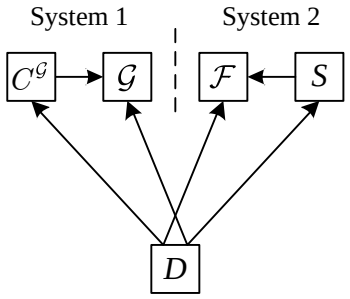


Figure 2: Distinguisher's view in the indifferentiability game

1. C vs. $\mathcal{F}$: since C is an iterative construction, it processes its input as a sequence of blocks which are sent to $\mathcal{G}$ in turn; however, this is not the case with $\mathcal{F}$ because $\mathcal{F}$ processes the whole message (no matter how long it is) independently at once.

2. $\mathcal{G}$ vs. S: it is clear that $\mathcal{G}$ will necessarily be aware of all queries sent to System 1 because C will eventually process its queries through $\mathcal{G}$, but S, on the other hand, cannot see queries sent to $\mathcal{F}$, and thus may not be aware of them.

It is, therefore, the job of the simulator S to account for these differences to resist D's distinguishing attacks. However, this is only possible if C is a good construction; in fact, there are cases where D will always succeed in exploiting these structural differences no matter how intelligent and efficient S is. To illustrate how D can typically exploit the differences between the systems and distinguishes them, we describe an attack against the Merkle-Damgård construction as reported in [33], which shows that the plain Merkle-Damgård construction is not indifferentiable from $\mathcal{RO}$; figure 3 illustrates the steps of the attack. Basically, D here exploits the fact that C processes its queries iteratively by calling $\mathcal{G}$, and that S cannot see queries sent to $\mathcal{F}$ (the differences listed above). The attack proceeds in three steps:

1. First, D sends the query m_1 to both C and $\mathcal{F}$ and receives $C(\mathcal{G}(IV, m_1)) = Z$ and $\mathcal{F}(m_1) = Z'$; the IV is hard coded at C and inserted automatically,

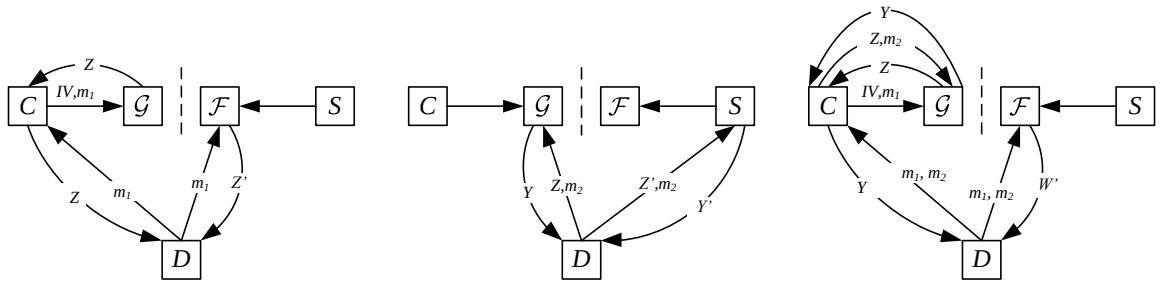
2. then, D sends the 2-block queries $Z||m_2$ and $Z'||m_2$ to $\mathcal{G}$ and S, respectively, and receives $\mathcal{G}(Z, m_2) = Y$, and $S(Z', m_2) = Y'$; here it will not make a difference whether S used F to get Y' or it generated it uniformly at random.

3. finally, D sends the 2-block query $m_1||m_2$ to C and $\mathcal{F}$ and receives $\mathcal{F}(m_1, m_2) = W$, and

$$C(m_1, m_2) = C(\mathcal{G}(\mathcal{G}(IV, m_1), m_2)) = C(\mathcal{G}(Z, m_2)) = Y$$

D then outputs 1 (i.e. success) in System 1 if $\mathcal{G}(Z, m_2) = C(m_1, m_2)$, and 0 otherwise (note that D decides on the success conditions which may be different when D interacts in different

settings). Similarly, D outputs 1 in System 2 if $S(Z', m_2) = \mathcal{F}(m_1, m_2)$. It is easy to see that D will always output 1 when interacting with System 1, but will output 0 with overwhelming probability when interacting with System 2 (this is because S cannot see m_1, so it can only guess it, but this has a low probability of success). In this case, D succeeds in its distinguishing game. In summary, proofs in the indifferentiability framework proceeds in two steps:



(a) Step 1 (b) Step 2 (c) Step 3

Figure 3: Indifferentiability attack against the Merkle-Damgård construction

1. First, we define a simulator S in System 2 (the Random System) to play the role of $\mathcal{G}$ in System 1 (the Real System), and whose relationship to the random oracle $\mathcal{F}$ mimics that of $\mathcal{G}$ to C.

2. Then, we prove that the view of a distinguisher D is similar when it has access to the random oracle $\mathcal{F}$ and the simulator S system (System 2), and the construction C and the ideal primitive $\mathcal{G}$ system (System 1), except with negligible probability. Formally, the advantage of an adversary D to success in the indifferentiability game (also called the Pseudorandom Oracle (PRO) game) is as follows:

$$\mathbf{Adv}^{\text{pro}}_{C}(D) = \left|\Pr\left[D^{C^{\mathcal{G}},\mathcal{G}} \to 1\right] - \Pr\left[D^{\mathcal{F},S^{\mathcal{F}}} \to 1\right]\right|$$

Game-playing The game-playing technique was first used in [63], then formalised in [17], and has been a popular technique for analysing cryptographic primitives ever since. This technique is usually used when we need to prove that two systems cannot be distinguished from each other. Let these systems be System 1 and System 2. We start with System 1 and write it as a *game* (pseudocode), we then introduce minor syntactical modification to the game and calculate the probability that a distinguisher will be able to distinguish between the original game and the one with the minor modification. The probability is usually calculated by introducing a flag `bad` that is originally set to false, and upper bounded by the probability that a distinguisher will set `bad` = true, where it succeeds in its distinguishing game. We then keep introducing similar minor syntactical changes to the games and track the probability that the distinguisher will set `bad` = true between every consecutive games. The simulation ends when we reach the game that is identical to System 2. The overall distinguishing advantage is then easily upper bounded by recalling all the probabilities of setting `bad` = true. This technique is primarily used in the indifferentiability and indistinguishability proofs.

Salvaging differentiable Constructions A serious problem arises when trying to use Coron's indifferentiability framework with hash functions based on mathematical primitives (such as those presented in section 9.2) because they are easily differentiable from $\mathcal{RO}$ due to the rigorous mathematical structure they exhibit (whereas $\mathcal{RO}$ are unstructured entities). However,

while it seems that this class of hash functions *cannot* be considered practical hash functions (since they are differentiable from $\mathcal{RO}$), Ristenpart and Shrimpton [90] pointed out (and proved) that such hash functions can be slightly modified to be indifferentiable from $\mathcal{RO}$. They proposed a construction called Mix-Compress-Mix (MCM) which basically wraps the hash function with two injective mixing steps to mix the input and the output of the hash function and *hide* its (mathematical) structure.

In [79], the authors adopted a slightly different approach which gives hope to the constructions that failed to be indifferentiable from $\mathcal{RO}$. Their approach involves introducing weaker $\mathcal{RO}$ variants and then proving that the constructions that failed to be indifferentiable from $\mathcal{RO}$ are indifferentiable from these weaker $\mathcal{RO}$ variants. Major cryptosystems, such as FDH, OAEP and RSA-KEM, are then secure under those constructions if they are secure in these $\mathcal{RO}$ variants. This approach was demonstrated on the Merkle-Damgård construction which was shown to be not indifferentiable from $\mathcal{RO}$ [33]. The authors proposed three $\mathcal{RO}$ variants as follows (in descending order from strongest to weakest): Leaky $\mathcal{RO}$ (LRO)[5], Traceable $\mathcal{RO}$ (TRO) and Extension attack simulatable $\mathcal{RO}$ (ERO). The authors proved that FDH is secure in LRO, OAEP is secure in TRO, and RSA-KEM is secure in ERO, then they proved that Merkle-Damgård is indifferentiable from LRO, TRO and ERO, which means that Merkle-Damgård is secure in FDH, OAEP and RSA-KEM.

3.2 Indistinguishability from PRF

In the keyed setting, a hash function is indistinguishable from Pseudorandom Function (PRF) if there is no adversary able to distinguish it from a random function. A random function is a function that has been chosen randomly based on a given domain and range; this does not imply that the output of the function should be random, the randomness here refers to the function selection process not the output of that function. Indeed, the function with constant output (e.g. always outputs 1) is a random function if it was selected randomly. Being a random function (or indistinguishable from one) is an idealisation of hash functions because when a hash function is modelled as a random function $H_K : \mathcal{K} \times \mathcal{M} \to \mathcal{Y}$, every key $K \in \mathcal{K}$ trigger the selection of a random function (member of the function family) that maps an input $M \in \mathcal{M}$ from the domain to a random output in the range $Y \in \mathcal{Y}$. Given such hash function family, it should be infeasible for an adversary A with black-box access to H_K to distinguish a randomly chosen member of H_K from a genuinely random function. Succinctly,

$$\mathbf{Adv}^{\mathrm{prf}}_{H_K}(\mathrm{A}) = \left|\Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathrm{A}^{H_K} = 1\right] - \Pr\left[R \xleftarrow{\$} Func(Dom, Rng) : \mathrm{A}^{R} = 1\right]\right|$$

where $Func(Dom, Rng)$ denotes the set of all functions mapping inputs from the domain Dom to outputs in the range Rng, with R a randomly chosen function from such a set. Basically, in both the PRO game (section 3.1) and the PRF game the adversary A's view should be similar when it interacts with a $\mathcal{RO}$ or PRF (random function), respectively, as it is when it interacts with the hash function H (A cannot distinguish between H and a genuine $\mathcal{RO}$ or PRF, except with negligible probability).

3.3 Unforgeability

MACs (Message Authentication Code) are popular cryptographic primitives used for authentication and integrity checks. One of the most established approaches of designing MACs is based on keyed hash functions, where the hash function is secretly keyed (and only legitimate parties possess the key). That is, both the sender and the receiver share a secret key (that is assumed

[5]The LRO was proposed earlier by Yoneyama [116], also called pub-RO, in [46].

to have been exchanged securely) which they use in conjunction with a hash function to check the integrity and authenticity of a particular message. Precisely, a MAC scheme consists of a tag generation algorithm and a tag verification algorithm. The sender uses the tag generation algorithm to generate a tag for a particular message (using its secret key), then it sends the message and the tag to the receiver. Once the receiver receives the message-tag pair (which may have been tampered with en-route), it runs them through the tag verification algorithm using its own secret key. If the message-tag pair is valid, the tag verification algorithm returns 1 and the message is authenticated, otherwise it returns 0 and the message is rejected. Note that here the tag verification algorithm basically just runs the tag generation algorithm on the message and the receiver's secret key, then compares the tag it generates with the tag the receiver received. Essentially, MACs are secret-key primitives, the counterpart of MACs in the public-key setting is digital signatures, which is yet another central applications of hash functions, but we do not address digital signatures in this paper. For a hash function to be a good MAC, it needs to be unforgeable. Given an adversary (forger) A, the formal unforgeability definition is as follows:

$$\mathbf{Adv}^{\mathrm{mac}}_{H_K}(A) = \Pr\left[K \xleftarrow{\$} \mathcal{K}, (M,T) \xleftarrow{\$} A^{H_K} : H_K(M) = T \wedge M \text{ is not queried}\right]$$

The advantage definition of MAC implies that it should be difficult for A (a forger) to find a valid pair of a message M and a tag T which can then be successfully validated by a MAC algorithm (this message-tag pair is then called a MAC forgery). A's aim is to find *any* valid pair of M and T (it is not about recovering the secret key K that the MAC algorithm used while generating T). That is, A builds the forgery pair (M', T') by repeatedly querying H_K with a set of adaptively chosen messages $M_1, \ldots, M_s$ and observes the returned tags $T_1, \ldots, T_s$, then A succeeds if it can generate a new message $M' \notin \{M_1, \ldots, M_s\}$ and a valid tag T' such that $H_K(M') = T'$.

3.4 Other Notions

It has also been suggested that hash functions should behave as a randomness extractor (extracting uniformly random bits from an input generated by imperfect randomness source) [44, 45]. However, in order for a hash function to possess randomness extraction properties, it may be necessary to make strong assumptions on the compression function that may not even be practically relevant.

Another notion is PrA (Pre-image Awareness) proposed by Dodis *et al.* in [45], which states that if an attacker can find a pre-image M of a previously published hash value Y, then it must have already known (was aware of) M; that is, a hash function is PrA if there is no adversary that can find a pre-image of a previously published hash value, unless it is already aware of that pre-image (Dodis *et al.* showed that strengthened Merkle-Damgård preserves PrA). A strengthened variants of PrA, called adaptive pre-image resistance [66], allows the adversary to make adaptive queries to the underlying compression function. It was shown that a collision resistant compression function that preserves the adaptive pre-image resistance property can yield a hash function indifferentiable from $\mathcal{RO}$.

4 Multi-Property-Preserving

One would naturally think that having a hash function provably preserving some strong security property (such as indifferentiability from $\mathcal{RO}$) is enough to imply a sufficient security margin. However, in [13] Bellare and Ristenpart refuted this assumption by providing counterexamples showing that the constructions proposed by Coron *et al.* in [33] are in fact *not* collision resistant

while they are still indifferentiable from $\mathcal{RO}$, a supposedly strong security notion[6]. Thus, the authors suggested that a reasonably secure hash function should be *multi-property-preserving* (MPP).

In [94], Rogaway and Shrimpton provided a formal discussion about the relations between collision resistance, pre-image resistance, second pre-image resistance and several variants of the latter two. They considered families of hash functions (keyed hash functions) while studying these properties, because the keyed setting is easier to formally analyse than the keyless setting [93]. These properties are CR, Pre, Sec, aPre, ePre, eSec, aSec, whose advantages are as follows:

$$\mathbf{Adv}^{\mathrm{cr}}_{H_K}(\mathrm{A}) = \Pr\left[\, K \xleftarrow{\$} \mathcal{K}; (M, M') \xleftarrow{\$} A(K) : M \neq M' \wedge H_K(M) = H_K(M') \,\right]$$

$$\mathbf{Adv}^{\mathrm{Pre}[m]}_{H_K}(\mathrm{A}) = \Pr\left[\begin{array}{l} K \xleftarrow{\$} \mathcal{K}; M \xleftarrow{\$} \{0,1\}^m; \\ Y \leftarrow H_K(M); M' \xleftarrow{\$} A(K, Y) : H_K(M') = Y \end{array}\right]$$

$$\mathbf{Adv}^{\mathrm{aPre}[m]}_{H_K}(\mathrm{A}) = \Pr\left[\begin{array}{l} (K, St) \xleftarrow{\$} A(); M \xleftarrow{\$} \{0,1\}^m; \\ Y \leftarrow H_K(M); M' \xleftarrow{\$} A(Y, St) : H_K(M') = Y \end{array}\right]$$

$$\mathbf{Adv}^{\mathrm{ePre}}_{H_K}(\mathrm{A}) = \Pr\left[\, (Y, St) \xleftarrow{\$} A(); K \xleftarrow{\$} \mathcal{K}; M' \xleftarrow{\$} A(K, St) : H_K(M') = Y \,\right]$$

$$\mathbf{Adv}^{\mathrm{Sec}[m]}_{H_K}(\mathrm{A}) = \Pr\left[\begin{array}{l} K \xleftarrow{\$} \mathcal{K}; M \xleftarrow{\$} \{0,1\}^m; \\ M' \xleftarrow{\$} A(K, M) : M \neq M' \wedge H_K(M) = H_K(M') \end{array}\right]$$

$$\mathbf{Adv}^{\mathrm{aSec}[m]}_{H_K}(\mathrm{A}) = \Pr\left[\begin{array}{l} (K, St) \xleftarrow{\$} A(); M \xleftarrow{\$} \{0,1\}^m; \\ M' \xleftarrow{\$} A(M, St) : M \neq M' \wedge H_K(M) = H_K(M') \end{array}\right]$$

$$\mathbf{Adv}^{\mathrm{eSec}[m]}_{H_K}(\mathrm{A}) = \Pr\left[\begin{array}{l} (M, St) \leftarrow A(); K \xleftarrow{\$} \mathcal{K}; \\ M' \xleftarrow{\$} A(K, St) : M \neq M' \wedge H_K(M) = H_K(M') \end{array}\right]$$

Let xxx $\in$ {Pre, Sec}, then saying that H_K is a-xxx means the hash function H_K is *always* xxx-resistant for a fixed key K and random challenge, while e-xxx means the hash function H_K is *everywhere* xxx-resistant for a fixed challenge and random key K. The challenge in Sec is the original message M (domain point) to which we need to find a second pre-image, while it is the hash value $H_K(M)$ (range point) in Pre which we need to invert to find M. Occasionally, the adversary A may return a state variable St, which contains information that the adversary may need in later stages of the attack (this is how extra information is usually modelled in formal definitions). For example, if A generates a key K, A may wish to keep track of any random choices he made during the generation of K, so A stores this information in St.

The eSec property is also called Target Collision Resistance (TCR) [16] which is, in turn, another name for the popular Universal One Way Hash Function (UOWHF)[7] notion of Naor and Young [77]. Strengthened variants of some of these properties were proposed in [56, 113,

[6]However, that does not degrade the importance of the indifferentiability as a property.

[7]In UOWHF (or TCR or eSec), an adversary A generates a message M, then given a random key K, A generates another message $M' \neq M$, such that $H_K(M) = H_K(M')$. A strengthened variant of TCR is eTCR where A aims at finding $M \neq M'$ such that $H_K(M) = H_{K'}(M')$ and $K \neq K'$.

89], namely s-CR, s-Sec, s-aSec, s-eSec, s-Pre, s-aPre. The authors argued that ePre cannot be strengthened because if it was strengthened, then there will always be a trivial adversary succeeding in the s-ePre game.

Backward Chaining Mode (BCM) proposed by Andreeva and Preneel in [8] preserves the three classical security properties of hash functions, namely CR, Pre and Sec. Similarly, in [7] Andreeva *et al.* proposed the ROX (Random Oracle XOR) construction which preserves seven properties: CR, Pre, ePre, aPre, Sec, eSec and aSec. However, later work by Reyhanitabar *et al.* in [88] showed that ROX is surprisingly not indifferentiable from $\mathcal{RO}$ even though ROX uses two $\mathcal{RO}$s in its padding algorithm. Moreover, in [14] Bellare and Ristenpart proposed the ESh (Enveloped Shoup) construction that preserves: CR, eSec, PRO-Pr, PRF-Pr and MAC, but later Reyhanitabar *et al.* [88] showed that ESh does not preserve Sec, aSec, Pre and aPre.

5 Cryptographic Proofs

There are two popular general approaches that proofs of cryptographic hash functions usually adopt, either constructing the proof in the standard model, or in the ideal model (regardless of whether the hash function was keyless or keyed). Proofs in the standard model assume the presence of primitives preserving/possessing standard (practical) properties, such as collision resistance, pre-image resistance, 2nd pre-image resistance etc. A standard model proof is then developed to argue that a hash function preserves such properties if it is given access to (or built from) primitives preserving those properties. For example, the hash function H is said to be collision resistant if we can construct a standard model proof that demonstrates the following:

> *If a hash function H^G has oracle access to a standard primitive G, and G is collision resistant, then H^G is also collision resistant.*

Such proofs are said to be constructed in the standard model. On the other hand, proofs in the ideal model assume the presence of ideal primitives and proceed by proving that if a hash function has access to (or built from) such primitives, it possesses a particular property (or set of properties). For example, a hash function H is said to possess the property xxx if a proof can be develop to argue about the following:

> *If a hash function $H^{\mathcal{G}}$ has oracle access to an ideal primitive $\mathcal{G}$, then $H^{\mathcal{G}}$ provably possesses the property* xxx*, or indifferentiable from $\mathcal{G}$.*

The ideal primitive $\mathcal{G}$ can be any idealistic component, such as an ideal permutation, but the most commonly used primitive in cryptography (in generally) is a random oracle, which we discussed fairly thoroughly in section 3.1. Thus, the Random Oracle Model (ROM) can be thought of as the most popular ideal model, but it is not the only one. In fact, in chapter **??**, we will develop our indifferentiability proof by adopting the Ideal Cipher Model (ICM), which assumes the presence of an ideal block-cipher since hash function are usually (directly or indirectly) built from block-ciphers.

While proofs in the standard model are more practically relevant, it is sometimes extremely difficult to carry out such proofs for for some schemes, potentially making the the ideal model the only (feasible) option. Clearly, nonetheless, proofs in the ideal model obviously provide weaker security guarantees because they assume the presence of ideal primitives that may not exist in practice. In both approaches (an in this thesis), the adversary is usually assumed to be have finite computational resources.

Part II
Hash Functions Design

6 Keyless vs. Keyed Hash Functions

Generally, hash functions are classified as keyless or keyed. Keyless hash functions accept a variable[8] length message M and produces a fixed length hash value, $H : \{0,1\}^* \rightarrow \{0,1\}^n$. Keyed hash functions, on the other hand, accept both a variable length message M and a fixed length key K to produce a fixed length hash value, $H_K : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^n$. Keyed hash functions can be further classified based on whether the key is private or public. Secretly keyed hash functions are usually used to build Message Authentication Codes (MAC), the canonical example is HMAC [78]; see section 3.3 for more information about MACs. If, however, the hash functions are publicly keyed, they are commonly known as dedicated-key hash functions [38, 14]. Hash functions designed in the dedicated-key setting are families of hash functions where individual member functions are indexed by different keys. In this setting, if a member of the hash function family was broken, this should have minimal effect on the other members of the same family (this is not the case in the keyless setting where a single attack against a function breaks the function entirely, e.g. [110, 111]). An obvious drawback of hash functions in the dedicated-key setting, however, is a degraded efficiency since in this case the function is required to process an extra input (the key) beside the message input.

In general, a hash function (keyed or keyless) is built out of two components: a compression function f and a construction H. The compression function is a function mapping a larger (but fixed) sized input to a smaller fixed sized output $f : \{1,0\}^m \rightarrow \{1,0\}^n$, where $m > n$. The construction is the way the compression function is repeatedly called to process a message; refer to table 1 for brief (informal) definitions of some hash functions terminology which will be used interchangeably throughout the paper; also see [32] for discussion about the lack of standard terminology and definitional consistency in the hash functions literature.

7 Iterative Hash Functions

When hash functions first emerged, it was realised that the most convenient way to hash a message is by first dividing it into several blocks and then iteratively and systematically processing these blocks. Today, this sequential hashing approach is still, by far, the most widely used, even with the advent of parallel processors (which, at least in principle, should have given advantage to the parallel hash functions). In the following subsections, we review some popular iterative hashing constructions and discuss how recent designs tried to fix weaknesses in earlier ones. The absence of a particular construction in the sections below does not imply that we disfavour it; indeed it is nearly impossible to be exhaustive in such a rapidly growing literature.

7.1 Merkle-Damgård Construction

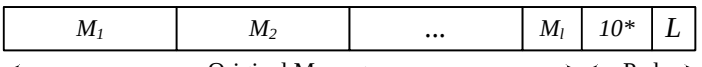
Most of today's popular hash functions, such as MD5 and SHA-1, are based on the infamous Merkle-Damgård construction (also called the cascade construction) proposed independently by Merkle [74] and Damgård [38] in 1989 (though, Damgård's construction was keyed while Merkle's

[8] The term *variable* in this context indicates that the length of the message is upper bounded by a large number of bits $M = \{0,1\}^{\leq \lambda}$ (e.g. $\lambda = 64$) that is sufficient to represent any message in practice. The term *arbitrary* [72], on the other hand, describes messages with infinite length, $M = \{0,1\}^*$. However, some authors use the two terms interchangeably. In this paper, unless stated otherwise, we will always use variable length messages, but for convenience, we will use the notation $\{0,1\}^*$.

Terminology	Informal Definition
Compression/Compressing Function	A standard building block of a hash function, with its domain larger than its range
Construction, Transform, Mode of Operation, Chaining Mode, Domain Extension Transform, Composition Scheme	An algorithm that systematically makes repeated calls to a building block(often a compression function) to hash a message.
Chaining Variable, Chaining Value, Intermediate Hash, Internal State	The output of a compression function to be used as input to the following compression function call.
Hash value, Final Hash Value, Hash Code, Hash Result, Hash, Digest	The final result of hashing a message, which is a fixed length string.

Table 1: Some Hash Function Terminology

was keyless). However, it appears that similar construction has previously been proposed by Rabin [87] in 1978, raising some controversy in whether it should be called Rabin's construction instead. Nevertheless, while Rabin did indeed propose this construction, it was Merkle and Damgård who formally proved that it is collision resistant if its underlying compression function is collision resistant. In the Merkle-Damgård construction, the message M is first divided into equally sized blocks, $M = M_1, M_2, \ldots, M_\ell$. If the message M fell over or below the block boundaries, it is padded. To be collision resistance, the length of the message is appended to the message after padding it, this is termed Merkle-Damgård *strengthening* (first coined by Lai and Massey in [65], though already proposed by Merkle [74] and Damgård [38]); figure 4 illustrates the padding algorithm, where L is a 64-bit encoding of the the length of the message[9] and m is the length of a single block. The message is then iterated repeatedly by calling a Fixed-Input-Length (FIL) compression function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ accepting two inputs: a message block M_i (of length m) and either an Initialisation Vector IV (when hashing the first block) or a chaining variable (which is the output of the previous f call), both of length n; figure 5 provides a depiction and a pseudocode of the Merkle-Damgård construction.



Algorithm $\text{Pad}_s(M)$

$d = M + 1 + 64 \mod m$

$M||1||0^d||\langle M\rangle_{64} \to \hat{M}$

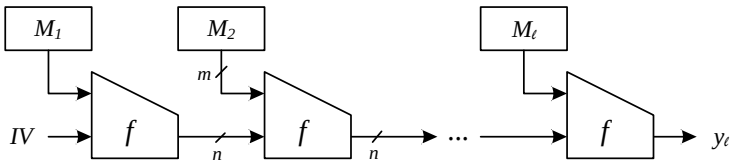
$\hat{M} \to M_1 \ldots M_\ell$

Figure 4: (Strengthened) Merkle-Damgård padding algorithm

7.2 Generic Attacks Against Merkle-Damgård

Eventually, several weaknesses were found in the Merkle-Damgård construction giving raise to a class of *generic* attacks that is applicable to any hash function based on the plain Merkle-Damgård construction. Note the difference between *generic* and *dedicated* attacks, where dedicated attacks exploit internal structures specific to a particular hash function and thus only affect

[9] While the padding algorithm illustrated in figure 4 is the most commonly used, it is not the only one. The Enveloped constructions such as ESh and CS (section 7.3) use different padding algorithms.



Algorithm MD^f

$M \rightarrow M_1 \dots M_\ell$
$y_0 = IV$
for $i = 1$ **to** ℓ **do**
  $y_i = f(M_i, y_{i-1})$
return y_ℓ

Figure 5: The Merkle-Damgård Construction

that hash function (e.g. the attacks against MD4, MD5 and SHA-1 by Wang *et al.* [110, 111] are dedicated attacks). Below we discuss four generic attacks against the Merkle-Damgård construction; even though the practical relevance of these attacks is not clear, they still demonstrate intrinsic structural weaknesses in the Merkle-Damgård construction (we will later show how variants of the Merkle-Damgård construction succeeded in thwarting some of these attacks).

The Extension Attack. It is often claimed that this attack was first reported by Ferguson and Schneier [51] in 2003 where it was described as a "surprisingly serious (and simple) flaw" in the Merkle-Damgård construction. However, it seems that the basic idea of this attack was discovered long before 2003 by Solo and Kent who called it the padding attack [107]. We present four variants of this attack as follows:

- *Collision Attack.* Suppose we have a message M with length $|M| = L$ that hashes to $H(M)$, then given any hash function $H(.)$ based on the Merkle-Damgård construction, a collision is trivially found as follows: $H(M||pad||x) = H(H(M)||x)$ where pad is the padding appended to the message M before being hashed and $|pad| = L \bmod m$, where m is the length of a single block in M, which is usually $|H(M)|$; note that $|pad|$ can indeed be 0 if the message was perfectly aligned at block boundaries. However, this attack does not consider Merkle-Damgård strengthening (appending the message length to the message before hashing it).

- *Second Collision Attack.* In this attack, a collision can easily be found by extending equally sized already colliding messages. That is, if we have $H(M) = H(N)$ while $M \neq N$ and $|M| = |N|$, a second collision can be obtained by extending M and N with an arbitrary string (suffix) S, $H(M||S) = H(N||S)$. This will work with Merkle-Damgård strengthening, but without strengthening, a second collision is even easier as the colliding messages no longer have to be equally sized.

- *Related Message Attack.* With Merkle-Damgård, one can easily compute a related/extended message M' for an unknown message M by only knowing L (length of M) and $H(M)$, that is, $H(M||L||x)$ is the hash of a message consisting of the original M and extended by a suffix $L||x$; again, since the attacker knows L, it is trivial to figure out how M has been padded before being hashed. This attack indeed does not affect collision resistance, but it shows that the Merkle-Damgård construction does not behave like a random oracle, which is a desirable property that hash functions should possess; see section 3.1.

The Multi-collision Attack. In [60], Joux showed that finding multiple collisions (more than two messages hashing to the same value) in a Merkle-Damgård hash function is not much harder than finding single collisions. In his multi-collision attack, Joux assumed access to a machine $\dot{C}$ that given an initial state, returns two colliding messages ($\dot{C}$ may use the birthday attack or any other attack exploiting weaknesses in the corresponding hash function). Figure 6 illustrates the attack.

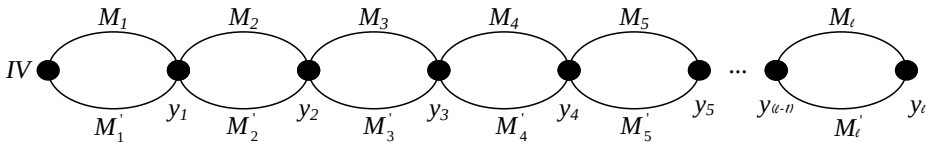


Figure 6: Multi-collision Attack

In figure 6, initially, the IV is sent to $\dot{C}$ which returns M_1 and M_1' colliding at y_1, then y_1, in turn, is sent to $\dot{C}$ which returns M_2 and M_2' colliding at y_2, this process continues until reaching y_ℓ as shown in figure 6. It is easy to see that any combination of the messages preceding y_ℓ will collide in y_ℓ. In fact, in the example in figure 6, there are 2^ℓ messages all colliding in y_ℓ and the cost of generating these collisions is only ℓ times the cost of generating single collisions.

2nd Pre-image Attack. This attack was first proposed by Dean in [41] and later generalised by Kelsey and Schneier in [62]. The attack assumes the existence of a set of expandable messages; these are messages of different lengths but produce the same intermediate hash value (chaining variable) given a particular IV. Expandable messages, however, do not produce the same final hash value due to Merkle-Damgård strengthening. These messages can easily be found if the hash function contains fixed points[10]. While not an intrinsic weakness of the Merkle-Damgård construction, fixed points can be found in many Merkle-Damgård implementations (e.g. SHA-1) because the compression functions are usually modelled as Davies-Meyer functions where the chaining variable input of the compression function is further XORed with its output. Let a Merkle-Damgård hash function be denoted by H, and suppose we have a set of such messages $E = E_1, \ldots, E_l$ which all produce an intermediate hash value y_E. Let $M = M_1, M_2 \ldots, M_\ell$ be a very long message consisting of ℓ blocks, and $C = c_1, \ldots, c_{\ell-1}$ be the set of all the intermediate hash values of M (for a message consisting of ℓ blocks, there are $\ell - 1$ intermediate hash values). Now, search for a block M_i in M, such that $f(y_E, M_i) \in C$. Suppose M_i is found (finding M_i has complexity less than 2^n since M is a very long message, where n is the length of the final hash value of H) and it matches c_j (the j-th intermediate hash value of M), now search E (the set of expandable messages) for a message E_s of length j-1 such that the number of blocks of $E_s||M_i$ is j. Let the original message M without its first j blocks be M', then $H(E_s||M_i||M') = H(M)$. This attack finds a 2nd pre-image for a message of size 2^k in $2^{n/2+1}+2^{n-k+1}$ steps rather than the expected 2^n. For example, using RIPEMD-160, it finds a 2nd pre-image for a 2^{60} byte message in around 2^{106} steps, rather than the expected 2^{160} steps, where RIPEMD-160 produces a hash digest of size 160 bits [62].

The Herding Attack. This attack is due to Kelsey and Kohno [61] and is closely related to the multi-collision and 2nd pre-image attacks discussed above. A typical scenario where this attack can be used is when an adversary commits to a hash value D (which is not random) that he makes public and claims (falsely) that he possesses knowledge of unknown events (events in the future) and that D is the hash of that knowledge. Later, when the corresponding events occur, the adversary tries to herd the (now publicly known) knowledge of those events to hash to D as he previously claimed. The attack proceeds in two phases:

- phase 1: construct a diamond structure and calculate the value D.
- phase 2: given prefix, find a suitable suffix and herd it to D through the diamond.

[10] A fixed point is found when two consecutive chaining values collide, that is $f(h_{i-1}, M_i) = h_i = h_{i-1}$, where f is a compression function.

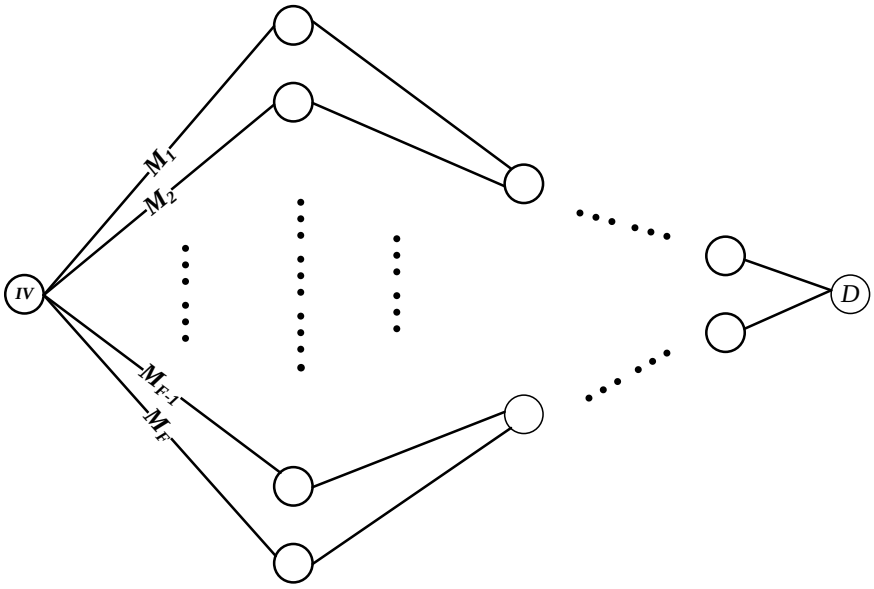


Figure 7: Diamond Structure

In phase 1, the attacker constructs a diamond structure as shown in figure 7, where the vertices are hash values and the edges are messages. If two messages meet in a vertex, they collide at that vertex. Initially, the attacker randomly generates an arbitrary large number of initial messages, $M_1, \ldots, M_F$, hashes them and tries to find collisions, then repeat until reaching the root of the diamond, D. Once the diamond is constructed, any path from the initial messages to D will hash to D. In phase 2, the attacker *herds* a given prefix P to hash to D as follows: first, the attacker searches for a suitable 1-block suffix S that if concatenated with P, it will produce a hash colliding with one of the hash values of the initial messages $H(M_i)$ where $i \in \{1, 2, \ldots, F\}$; for 2^k initial messages, 2^{n-k} trials are required to find such a suffix (where n is the length of the final hash). Once a match is found, P, S and the sequence of messages from the matching $H(M_i)$ to D are concatenated, and this whole string will eventually hash to D. The herding attack was recently extended to non-Merkle-Damgård constructions [48, 3]. A more detailed complexity analysis for the herding and diamond-based attacks are presented in [104], which point out a flaw in the construction proposed in [61] to produce a diamond structure, and provide computational complexity analysis for constructing the diamond. To resist this attack, hash functions should possess the Chosen Target Forced Prefix (CTFP) pre-image resistance property; see section 2.4.

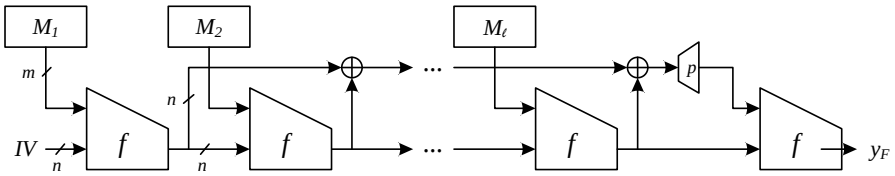
7.3 Variants of Merkle-Damgård

The discovery of the weaknesses reported in section 7.2 drove the research community to propose modified variants of the Merkle-Damgård construction that patch such weaknesses. In this section, we present a few examples of both keyless and keyed constructions (most of which trade off efficiency for security). Note that some of these constructions use different padding algorithms than the standard one in figure 4.

Wide and Double Pipe. One of the earliest proposals to enhance the Merkle-Damgård construction is the wide/double pipe by Lucks [68] who showed that increasing the size of the internal state (i.e. the chaining variables) to become larger than the size of the final hash value, would significantly improve the security of the hash function. This modification clearly thwarts

the extension attack since in the wide/double pipe the final hash value is truncated, so in order to append an extension, the unknown discarded bits have to be guessed, which is clearly difficult if the number of the discarded bits is non-trivial. Furthermore, by increasing the size of the internal state, finding collisions for the compression function becomes even harder, which complicates the other generic attacks. An obvious drawback of the wide/double pipe, however, is a degraded efficiency as the compression function now has larger input/output while keeping the hashing rate constant (the size of the compression function input corresponding to a message block is fixed) since the chaining variable input is increased. Also, adapting existing hash functions for the wide/double pipe may be difficult since it might be the only reasonable way to increase the internal state is to use multiple compression function calls in parallel for every iteration. Recently, Yasuda [114] adopted a slightly modified variant of the double pipe construction and proved its unforgeability beyond the birthday barrier.

The 3C Construction. Another variant of the Merkle-Damgård construction is the 3C construction [53] which basically maintains a variable containing a value produced by repeatedly XORing the chaining variables while hashing a message; this variable is then processed in an extra finalisation call to the compression function. Figure 8 illustrates the 3C construction. An enhanced variant of 3C is 3C+ which uses extra memory, but makes finding *multi-block* collisions more difficult (not to be confused with multi-collision attack, see section 2). However, in [59], it was shown that both 3C and 3C+ are indeed susceptible for multi-block attack; this was demonstrated using a recent attack against MD5 that was found to be applicable for both the plain Merkle-Damgård and 3C/3C+. Also, 3C does not seem to resist multi-collision attacks since the internal states are not affected by the modification introduced in 3C.



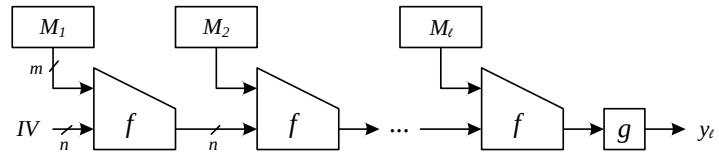
Algorithm $3C^f$

$M \to M_1 \dots M_\ell$
$y_0 = IV, t = 0$
for $i = 1$ **to** ℓ **do**
  $y_i = f(M_i, y_{i-1})$
  $t = t \oplus y_{i-1}$
return
  $y_F = f(P(t), y_\ell)$

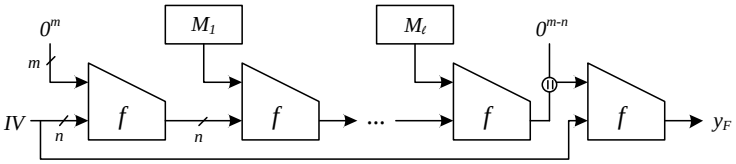
Figure 8: The 3C Construction

The Prefix Free, Chop, NMAC and HMAC Constructions. Several constructions were proposed by Coron *et al.* in [33] as immediate fixes to the Merkle-Damgård construction after showing that the latter is not indifferentiable from $\mathcal{RO}$ (see section 3.1 for details about the indifferentiability notion). However, Bellare and Ristenpart [13] later showed that even though these constructions are indifferentiable from $\mathcal{RO}$, they are not collision resistant. The prefix-free construction does not modify the Merkle-Damgård construction, instead it modifies the padding algorithm to make sure that the message is prefix free. One way to do this is by prepending or appending the length of the whole message to every message block. However, beside wasting a few bits to represent the length of the message in every block and so degrading the efficiency, this obviously does not work well with streaming applications (where the length of the message is not known beforehand). The chop construction basically removes a non-trivial number of bits from the final hash value. This, while it solves the indifferentiability issue, unfortunately lowers the security bounds of the hash function. In NMAC, an independent function g is applied to the

output of the last application of the compression function, while in HMAC an extra compression function call is introduced. The NMAC and HMAC constructions are illustrated in figure 2.



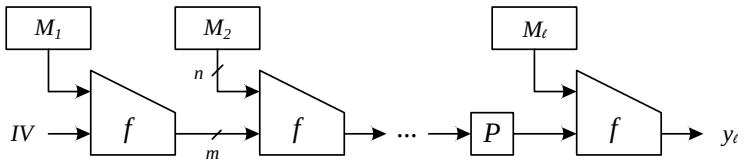
Algorithm NMAC^{f,g}
$M \rightarrow M_1 \dots M_\ell$
$y_0 = IV$
for $i = 1$ **to** ℓ **do**
$y_i = f(M_i, y_{i-1})$
return $y_F = g(y_\ell)$



Algorithm HMAC^{f}
$M \rightarrow M_1 \dots M_\ell$
$M_0 = 0^m, y_0 = f(M_0, IV)$
for $i = 1$ **to** ℓ **do**
$y_i = f(M_i, y_{i-1})$
return
$y_F = f(y_\ell || 0^{m-n}, IV)$

Table 2: The NMAC and HMAC Constructions

The Merkle-Damgård with Permutation. In [58] Hirose *et al.* proposed the Merkle-Damgård with Permutation (MDP) construction which introduces very minor modification to the plain Merkle-Damgård. The only difference between the plain Merkle-Damgård and MDP is that in MDP the chaining variable input of the last compression function is permuted. The authors proved that MDP is indifferentiable from $\mathcal{RO}$ while the collision resistance of MDP follows trivially from the collision resistance of the Merkle-Damgård construction as the former introduces minimal changes to the latter. The authors also discussed the security of possible simple MAC constructions based on MDP. However, although with such a simple modification, the authors succeeded in proving a significant security gain, MDP seems to be able to thwart only the extension attack, but not other Merkle-Damgård generic attacks. Also, recently it was shown that MDP is neither pre-image nor 2nd pre-image resistant [4]. Figure 9 illustrates MDP, where $\pi(.)$ is a permutation function.

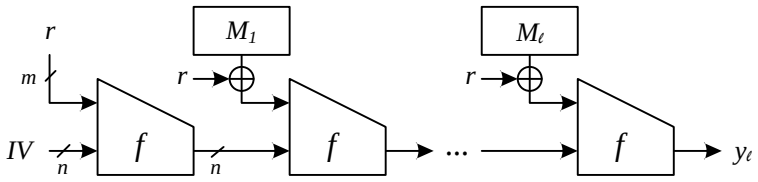


Algorithm MDP^{f}
$M \rightarrow M_1 \dots M_\ell$
$y_0 = IV$
for $i = 1$ **to** $\ell - 1$ **do**
$y_i = f(M_i, y_{i-1})$
return $y_\ell = f(\pi(y_{\ell-1}), M_\ell)$

Figure 9: The MDP (Merkle-Damgård with Permutation) Construction

Randomized Hashing. Randomized hashing [56] is not quite a variant of Merkle-Damgård, instead it is a generic fix that can be applied on any construction (including Merkle-Damgård). In randomized hashing the input of the hash function is randomised using a *salt*, leaving the construction unmodified, figure 10 illustrates the RMX transform [55], which is an instantiation of the randomized hashing paradigm. The authors claim that randomized hashing will strengthen any hash function, even the weakest ones. Randomized hashing was originally proposed for digital signatures where a message M is first randomised with a salt r to produce a randomised

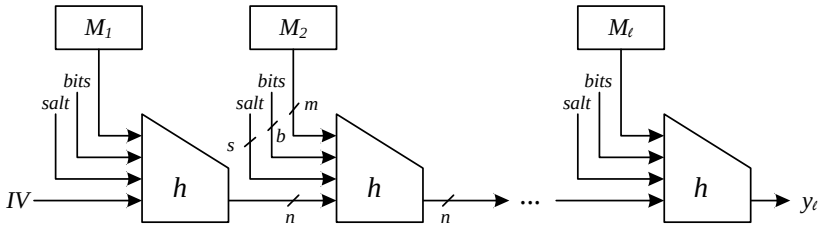
message M'. A digital signature sig is then generated from M'. The original message M, the salt r and the signature sig are then sent to the verifier. When the verifier receives these parameters, it first randomises M with r to produce M' and carries out standard signature verification using M' and sig. Randomized hashing for digital signatures are standardised by NIST in [40].



Algorithm RMX_r^f
$M \to M_1 \ldots M_\ell$
$y_0 = f(r, IV)$
for $i = 1$ **to** ℓ **do**
  $y_i = f(M_i \oplus r, y_{i-1})$
return y_ℓ

Figure 10: The RMX Construction

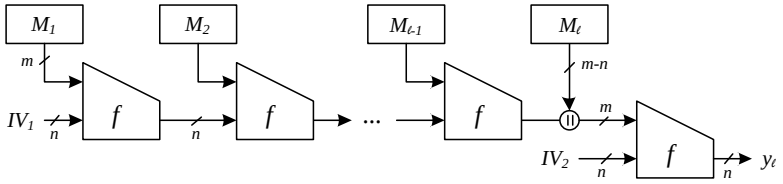
HAIFA Framework. HAsh Iterative FrAmework (HAIFA) is a modified Merkle-Damgård construction proposed by Dunkelman and Biham [47], see figure 11 for an illustration. HAIFA modifies Merkle-Damgård by introducing extra input parameters to the compression function. These are: a *salt* value (used as a key to create families of hash functions—if only one hash function is needed, the salt is set to 0), and the number of *bits* hashed so far, which thwarts many of the generic attacks against the plain Merkle-Damgård construction since the input to every compression function call becomes (with high probability) unique and highly dependent on where the compression function call is made through the hashing chain. In fact, HAIFA can be considered a dedicated-key hash function [14]. The idea of adding additional input parameters to the compression function has been previously proposed by Rivest through a process called *dithering* [91]; though a second pre-image attack against dithered hash functions was reported by Andreeva *et al.* in [2]. An obvious drawback of HAIFA is efficiency degradation since the compression function now has more input parameters to process. Furthermore, HAIFA cannot be (easily) used to patch existing Merkle-Damgård based hash functions because a compression function designed for the Merkle-Damgård construction would not naturally accommodate the extra HAIFA parameter inputs.



Algorithm HAIFA_s^h
$M \to M_1 \ldots M_\ell$
$y_0 = IV$
for $i = 1$ **to** ℓ **do**
  $y_i = h(M_i, y_{i-1}, b_i, s)$
return y_ℓ

Figure 11: The HAIFA Framework

Enveloped Merkle-Damgård. The Enveloped Merkle-Damgård (EMD) construction was proposed in [13] by Bellare and Ristenpart when they were introducing their multi-property-preserving notion, where they recommend that a particular hashing scheme should preserve multiple properties at the same time. This stemmed from the fact that they were able to prove the four constructions proposed by Coron *et al.* in [33] are not collision resistant while still being indifferentiable from $\mathcal{RO}$. Bellare and Ristenpart showed that EMD preserves collision resistance, indifferentiability from $\mathcal{RO}$ and indistinguishability from Pseudorandom Function (PRF). Figure 12 illustrates EMD.

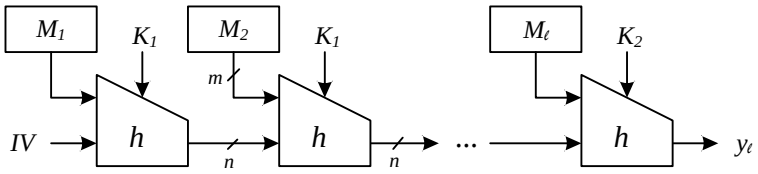


Algorithm $\text{EMD}^{f}_{IV_1,IV_2}$

$M \rightarrow M_1 \ldots M_\ell$
$y_0 = IV_1$
for $i = 1$ **to** $\ell - 1$ **do**
  $y_i = f(M_i, y_{i-1})$
return
  $y_\ell = f(y_{i-\ell}||M_\ell, IV_2)$

Figure 12: The EMD (Enveloped Merkle-Damgård) Construction

Nested Iteration. An and Bellare proposed the Nested Iteration (NI) mode of operation while they were proving that the Merkle-Damgård construction can be used to construct a Variable-Input-Length (VIL) MAC from a Fixed Input Length (FIL) MAC. NI is basically a keyed variant of the Merkle-Damgård construction making use of two keys $k_1, k_2 \in \{0,1\}^k$. Figure 13 illustrates the NI construction. Beside being unforgeable, Bellare and Ristenpart later proved in [14] that NI is also indistinguishable from PRF, indifferentiable from $\mathcal{RO}$, and if strengthening was used, NI is also collision resistant. However, neither NI nor its strengthened variant is target collision resistant (TCR); see section 4 for discussion about the TCR property.

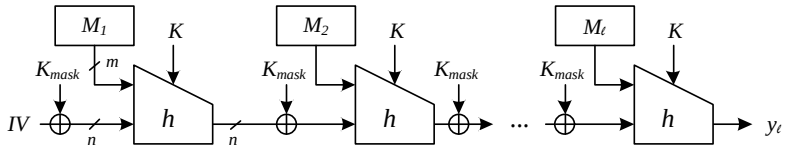


Algorithm $\text{NI}^{h}_{K_1,K_2}$

$M \rightarrow M_1 \ldots M_\ell$
$y_0 = IV$
for $i = 1$ **to** $\ell - 1$ **do**
  $y_i = h(K_1, M_i, y_{i-1})$
return $y_\ell = h(K_2, M_\ell, y_{\ell-1})$

Figure 13: The NI (Nested Iteration) Construction

Shoup (Sh) Construction. In [99], Shoup proposed an elegant keyed construction. In addition to the key input of the compression function, the chaining variables of every compression function iteration in Sh is further XORed with a key mask; figure 14 illustrates the Sh construction. A variant of the Sh construction has been proposed by Bellare and Ristenpart in [14] that makes the last compression function call a *wrapping* call (this last application of the compression function is called an *envelop*). Thus, this variant is called the Envelop Shop (ESh), which has been proven to preserve five important properties, namely: collision resistance, unforgeability, indifferentiability from $\mathcal{RO}$, indistinguishability from PRF and TCR. In [88] Reyhanitabar *et al.* further showed that ESh preserves the ePre property but not Sec, aSec, Pre and aPre (for information about these properties, see section 4 and/or [94]).

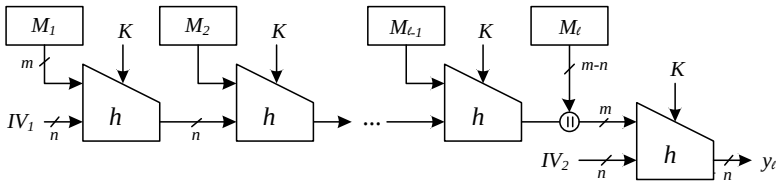


Algorithm $\text{Sh}^{h}_{K,K^{mask}}$

$M \rightarrow M_1 \ldots M_\ell$
$y_0 = IV$
for $i = 1$ **to** $\ell - 1$ **do**
  $y_i = h(y_{i-1} \oplus K^{mask}, M_i)$
return y_ℓ

Figure 14: The Shoup Construction

Chaining Shift. The Chaining Shift (CS) construction CS: $\{0,1\}^k \times \{0,1\}^{n+n} \times \{0,1\}^* \rightarrow \{0,1\}^n$ was proposed by Maurer and Sjödin in [72] as a more efficient solution than the NI construction for constructing AIL-MAC from FIL-MAC; figure 15 depicts the CS construction, which uses a FIL compression function $f : \{0,1\}^{m+n} \rightarrow \{0,1\}^n$. The CS construction was shown to be unforgeable [72], indistinguishable from PRF [14], indifferentiable from $\mathcal{RO}$ [13], and the strengthened variant of it (with strengthened padding) is collision resistant. Maurer and Sjödin have also simultaneously proposed the Chaining Rotate (CR) construction, which is similar to CS.



Algorithm $\mathrm{CS}^h_{K,IV_1,IV_2}$

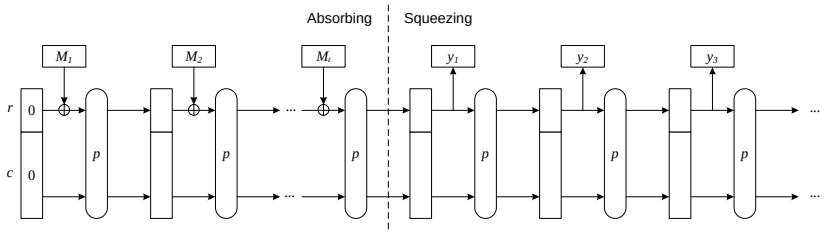
$M \rightarrow M_1 \ldots M_\ell$
$y_0 = IV$
for $i = 1$ **to** $\ell - 1$ **do**
  $y_i = h(y_{i-1}, M_i)$
return
  $y_\ell = h(IV_2 || y_{\ell-1}, M_\ell)$

Figure 15: The CS (Chaining Shift) Construction

7.4 Sponge Construction

Based on totally different design principles than Merkle-Damgård's, the Sponge construction is a newly proposed and promising hashing construction [21]. Basically, sponge hashing proceeds in two phases, the absorbing phase and the squeezing phase (and hence its name). The sponge operates on a fixed-length state $b = \{0,1\}^{r+c}$, composed of r bits (called bit-rate) and c bits (called capacity), through a function $p : \{0,1\}^{r+c} \rightarrow \{0,1\}^{r+c}$ which produces a transformation or permutation of b. In the absorbing phase, the message is divided into r-bit blocks (padded if necessary) and each block is XORed with the r part of b (initially, $b = 0^{r+c}$), p then iteratively processes b until all blocks are exhausted. In the squeezing phase, the state continues to be transformed/permuted by p but this time the r parts of the states are returned at every iteration as output blocks. Since the sponge construction supports variable length output, the user chooses the length of the final hash value which determines how many of the returned blocks in the squeezing phase need to be returned. Figure 16 illustrates the sponge construction. An example of a hash function based on the sponge construction is Keccak [23] which has recently been selected (along with 4 others) to advance to the final stage of the SHA-3 competition. Recently, Andreeva *et al.* introduced a generalisation of the sponge functions, which they call "The Parazoa Family" [6].

Although still considered an iterative construction, the sponge is completely different from the Merkle-Damgård construction; which obviously means that the generic attacks discussed in section 7.2 are not applicable. Moreover, the sponge construction has been proven to be indifferentiable from $\mathcal{RO}$ in [22]. However, that does not mean that the sponge construction is not susceptible to other kinds of attacks, it is just that (at the time of writing) such attacks have not been discovered yet. Recently, Gorski *et al.* [54] showed that hash functions based on the sponge framework may be susceptible to slide attack. On the other hand, an obvious disadvantage of sponges is that their relatively large state slows down the full diffusion of bits, hence, the sponge construction may be more suitable for hashing large messages.



Algorithm $Spng_n^p$
$M \rightarrow M_1 \ldots M_\ell$
$r = 0, c = 0$
for $i = 1$ **to** ℓ **do**
  $p(r \oplus M_i, c) = (r, c)$
for $i = 1$ **to** n **do**
  $Y = Y||r$
  $p(r, c)$
return Y

Figure 16: The Sponge Construction

8 Tree-based Hash Functions

Figure 17 illustrates a typical tree-based hashing construction. This is the most parallelisable class of constructions and is mainly suited for multi-core platforms where multiple processors can independently operate on different parts of the message simultaneously. An early tree-based mode of operation was proposed by Damgård [38] which was later optimised by Sarkar and Scellenberg [98], and Pal and Sarkar [81]. Similarly, Bellare and Rogaway [16] used a tree-based approach to build UOWHFs (Universal One Way Hash Functions) [77] which, although weaker than collision-resistant hash functions, are suitable for some applications. More recent works for building tree-based UOWHFs include [67] and [97]. In [12] Bellare and Micciancio proposed the *randomize-and-combine* paradigm, where the message is split into blocks, randomised individually and finally combined by an operation such as XOR (but XOR-based combiners are broken by a linear algebra attack on long messages [69]). Although this structure was originally proposed to build *incremental* functions[11], it can be thought of as a 2-level tree and can still be parallelised since the randomisation process of the individual blocks are independent (i.e. can be carried out by different threads/processors). Tree-based constructions are slightly less popular than the iterative ones because they are not as suitable for low-end platforms such as smart cards and RFID, which limits their utility. Skein [50] and MD6 [92] hash functions (SHA-3 candidates) provide a tree hashing mode beside the conventional iterative mode.

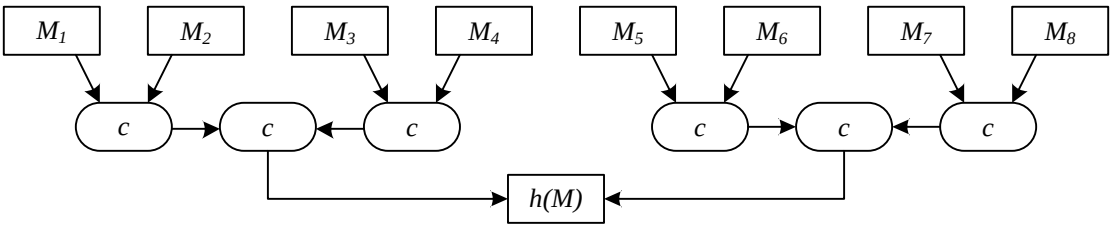


Figure 17: Sample tree construction

9 Compression Functions

Some hash functions are built from scratch such as MD5 and SHA-1, these are called dedicated hash functions. Others are built based on existing cryptographic or mathematical components that were not originally designed to be used for hashing but could be *tailored* to. So far, we

[11] An incremental function can efficiently *update* the digest of a previously hashed message to reflect any changes without having to re-hash the whole message.

have not explicitly discussed how to construct compression functions because when we design a hash function, one would assume the presence of a "good" compression function and design a construction accordingly. This obviously becomes an issue in the implementation phase. Thus, some proposals were exclusively concerned with building a compression function that preserves some property X and then adopt a suitable construction that is provably preserves X if the underlying compression function also preserves X (e.g. the Merkle-Damgård construction is collision resistant if the underlying compression function is also collision resistant).

9.1 Hash Functions Based on Block and Stream Ciphers

Building hash functions based on block ciphers is the most popular and established approach. In this approach, the compression function is a block-cipher with its two inputs representing a message block and a key. Preneel, Govaerts and Vandewalle [86] studied the 64 possible ways of constructing hash functions from a block-cipher $E : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$. These 64 constructions are sometimes called PGV constructions after the authors' initials who used an attack-based[12] analysis approach to study the security of these constructions. It was then reported that 12 out of the 64 PGV constructions are collision resistant, but later Black *et al.* [25] showed (using proof-based approach this time) that another 8 PGV constructions are also collision resistant if they were properly iterated, even if their underlying compression functions are not collision resistant. The most widely adopted construction of these 20 PGV construction is the one attributed to Davies and Meyer [73]: $y_i = f(h_{i-1}, M_i) \oplus y_{i-1}$, where y_{i-1} and M_i are the input of the compression function f, and y_i is its output. Another popular PGV construction is the Matyas-Meyer-Oseas construction [25], which is the opposite of the Davies-Meyer one. In Matyas-Meyer-Oseas, the output of the compression function y_i is further XORed with the message block input M_i (rather than the chaining variable y_{i-1} in Davies-Meyer). Further analyses and proofs of the collision resistance and pre-image resistance of these PGV constructions in the ideal cipher model can be found in [49] and [102].

Although PGV functions are provably secure, they are inefficient because the key (which represents the message block input of the compression function) is changed with every compression function call, and this is undesirable with block-ciphers since changing the key rapidly requires huge amount of computation (due to key setup). Thus, another approach is to use fixed-key block-cipher based compression functions [24, 95, 100, 101]. In this approach, a small non-empty set of keys are fixed and used for the block-cipher (when called by the compression function), while wrapping the block-cipher with other arbitrary functions to process the other compression function's input that was previously used as a key (which is now fixed). However, Black *et al.* [24] proved that such construction, making a single call to the fixed-key block-cipher, although efficient, cannot be collision resistant.

An inherent problem with designing hash functions based on block-ciphers is that block ciphers usually have small block size (e.g. 128 bit) which is insufficient to maintain an acceptable hash function security[13], unless the result of the hash function can be expanded, which proved to be even more difficult. A particularly interesting solution to this dilemma is designing double block length (DBL) compression functions where the compression function outputs double the size of the underlying block-cipher [76, 57]. Clearly, however, DBL based hash functions still scarify some efficiency.

Although the stream-cipher based approach is less popular than the block-cipher based approach, in the recent SHA-3 competition, some of the successful second round candidates were based on stream-ciphers (e.g. CubeHash [19]). The main differences between block-cipher-based

[12] Constructions that resisted the authors' attacks were deemed secure.

[13] For a block size of 128 bit, a collision can be found in 2^{64} operations due to the birthday attack, which may be at the edge of feasibility with the recent supercomputers processing technologies.

and stream-cipher-based hash functions are the size of the block and the number of rounds. In block-cipher-based, the message blocks are usually large, and iterated a small number of rounds, while in stream-cipher-based, the block size is small, with more rounds. Thus, in block-cipher-based, a good compression function is necessary but in stream-cipher-based, even a weak compression function may provide sufficient security.

9.2 Hash Functions Based on Mathematical Problems

The majority of today's well known hash functions process the message by mixing its bits in such a way that fulfils the various security and statistical requirements, but their security cannot usually be mathematically proven since they are not based on mathematical models. Provably secure cryptographic hash functions, on the other hand, is a class of hash functions that are based on mathematical problems where a rigorous mathematical proof can be derived such that it reduces breaking these hash functions to finding a solution to some hard mathematical problems. Examples of such hash functions include: hash functions based on the discrete logarithm problem [26, 30], hash functions based on the factorisation problem [31], hah functions based on finding cycles in expander graphs [29], etc. Designing several cryptographic primitives, including hash functions, based on the Knapsack problem[14] was more popular during the 90's. Although these schemes had good software and hardware performance, most of them are broken [84] which made knapsack based design approach less attractive. An example of such hash functions is the one proposed by Damgård [38] based on additive knapsack which was cryptanalysed in [82], and LPS hash function [29] based on a multiplicative knapsack, which was cryptanalysed in [106].

Another example of provably secure hash functions is syndrome-based hash functions [10, 11] which are based on an NP-complete problem known as *Regular Syndrome Decoding*. However, even though syndrome-based hash functions are provably secure, cryptanalytic results were published against several versions of these functions [34, 96, 52]. A recent syndrome based hash function is FSB [9] which was submitted to the SHA-3 competition, but failed to progress to round 2 of the competition, mostly due to its slow performance and excessive memory consumption, which seems to be the case with most hash functions based on mathematical problems (e.g. while the authors of DAKOTA [39] claim that it performs better than many other modular arithmetic-based hash functions, it is still approximately 18 times slower than SHA-256). Recently, Bernstein *et al.* [20] proposed an enhanced variant of FSB, called RFSB (Really fast FSB), and claimed that it runs at 13.62 cycles/byte, which is faster than SHA-2.

9.3 Other Approaches

Occasionally, attempts were made to adopt less common approaches when designing hash functions, most of which haven not attracted much interest. In this section, we discuss two such approaches: chaos-based and cellular automata based hash functions.

Chaos-based Hash Functions. Chaos theory is the mathematical representation of dynamic systems. These systems possess many desirable properties that suit the requirements of hash functions. For example, chaotic systems are very sensitive to changes in their initial values, potentially fulfiling the desirable hash function property requiring the output of the hash function to be highly sensitive to changes in its input; this phenomena is called the *avalanche effect* (also called butterfly effect in the chaos theory literature). Moreover, chaotic systems are one way functions and unpredictable. Hash functions based on chaos theory use *chaotic maps*,

[14] Knapsack is a famous combinatorics problem, where it is required to optimise the selection of items $d_1, \ldots, d_n$, where each item has a value v_i and a weight w_i, in such a way that the sum of the values $\sum_{i=1}^{n} v_i$ is maximised while keeping the sum of the weights $\sum_{i=1}^{n} w_i$ less than a particular threshold T.

which are functions that exhibit a particular chaotic behaviour; examples of these maps include: logistic map [70], tent map [115], and cat map [42]. Unfortunately, most chaos-based hash functions suffer from poor efficiency due to their inherent complex structure, which makes them unattractive as a practical approach for building hash functions.

Cellular Automata-based Hash Functions. Cellular Automata (CA) are discrete time models consisting of collections of cells organised in a grid, and each cell has a current state. The states of the cells evolve over time depending on their current states and the states of the neighbouring cells. CA were originally used by von Neumann [108] while he was studying self-reproducing systems and then popularised by Wolfram's substantial work in this area [112] who observed that based on simple rules, very complex behaviours can be obtained. Damgård was the first to propose a hash function based on CA [38], but his proposal was cryptanalysed by Daemen *et al.* [35] who, in turn, proposed another CA-based hash function, called CellHash. The same authors later proposed SubHash [36] which is an improved version of CellHash; both CellHash and SubHash are hardware-oriented and were cryptanalysed in [28]. Another hash function based on CA was proposed by Mihaljevie *et al.* [75].

10 Conclusion and Summary

Research in cryptographic hash function has recently witnessed an unprecedented spike of interest. According to [83], around 50-60 hash functions were available in 1993, followed by at least 30-40 others developed since then [85], in addition to the 64 SHA-3 submissions in 2007. This has lead to a rapidly expanding literature that we tried to survey in this paper. We discussed (both formally and informally) the most popular hash functions security properties and notions and showed how these requirements have influenced the design of hash functions over the years. In the first part of this paper, we elaborated on the three classical notions of collision resistance, pre-image resistance and 2nd pre-image resistance. We then provided a lengthy discussion on the indifferentiability from random oracle framework and showed how proofs in this framework are structured. In the keyed setting, a hash function should also be indistinguishable from a pseudorandom function, and, further, be unforgeable when used as a MAC. We talked about the multi-property-preserving (MPP) paradigm, where hash functions preserve multiple properties simultaneously; we also discuss a few examples of constructions from the literature exhibiting the MPP approach.

In the second part of this paper, we provided a thorough discussion of the state of art of hash functions design. Roughly speaking, hash functions can either be keyless or keyed. Each class has different applications and is based on different design principles. Hash functions can also be classified as iterative or parallel. While iterative functions are indeed the most common, parallelisable hash functions are increasingly being popularised with the rapid advent of parallel systems. We provided a lengthy discussion about the popular Merkle-Damgård construction, how it fell prey to several generic attack, and what approaches were adopted to strengthen it. Finally, we also discussed how compression functions are being designed and what approaches are adopted.

References

[1] Saif Al-Kuwari, James Davenport, and Russell Bradford. Cryptographic Hash Functions: Recent Design Trends and Security Notions. In *Short Paper Proceedings of Inscrypt '10*, pages 133–150. Science Press of China, 2010. (Full version available at `opus.bath.ac.uk/20815`). 1

[2] Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan Hoch, John Kelsey, Adi Shamir, and Sebastien Zimmer. Second Preimage Attacks on Dithered Hash Functions. In *Eurocrypt '08*, volume 4965 of *LNCS*, pages 270–288. Springer-Verlag, 2008. 22

[3] Elena Andreeva, Orr Dunkelman, Charles Bouillaguet, and John Kelsey. Herding, Second Preimage and Trojan Message Attacks Beyond Merkle-Damgård. In *SAC '09*, volume 5867 of *LNCS*, pages 393–414. Springer-Verlag, 2009. 19

[4] Elena Andreeva, Bart Mennink, and Bart Preneel. Security Properties of Domain Extenders for Cryptographic Hash Functions. *Journal of Information Processing Systems*, 6(4):453–480, 2010. 21

[5] Elena Andreeva, Bart Mennink, and Bart Preneel. Security Reductions of the Second Round SHA-3 Candidates. In *ISC '11*, volume 6531 of *LNCS*, pages 39–53. Springer-Verlag, 2011. 3

[6] Elena Andreeva, Bart Mennink, and Bart Preneel. *The Parazoa Family: Generalizing the Sponge Hash Functions*, 2011. (eprint.iacr.org/2011/028). 24

[7] Elena Andreeva, Gregory Neven, Bart Preneel, and Thomas Shrimpton. Seven-Property-Preserving Iterated Hashing: ROX. In *Asiacrypt '07*, volume 4833 of *LNCS*, pages 130–146. Springer-Verlag, 2007. 14

[8] Elena Andreeva and Bart Preneel. A Three-Property-Secure Hash Function. In *SAC '09*, volume 5381 of *LNCS*, pages 228–244. Springer-Verlag, 2009. 14

[9] Daniel Augot, Matthieu Finiasz, Philippe Gaborit, Stephane Manuel, and Nicolas Sendrier. *SHA-3 Proposal: FSB*, 2008. (www-rocq.inria.fr/secret/CBCrypto). 27

[10] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. *A Fast Provably Secure Cryptographic Hash Function*, 2003. (eprint.iacr.org/2003/230). 27

[11] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A Family of Fast Syndrome Based Cryptographic Hash Functions. In *Mycrypt '05*, volume 3715 of *LNCS*, pages 64–83. Springer-Verlag, 2005. 27

[12] Mihir Bellare and Daniele Micciancio. A New Paradigm for Collision-free Hashing: Incrementality at Reduced Cost. In *Eurocrypt '97*, volume 1233 of *LNCS*, pages 163–192. Springer-Verlag, 1997. 25

[13] Mihir Bellare and Thomas Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In *Asiacrypt '06*, volume 4284 of *LNCS*, pages 299–314. Springer-Verlag, 2006. 12, 20, 22, 24

[14] Mihir Bellare and Thomas Ristenpart. Hash Functions in the Dedicated-Key Setting: Design Choices and MPP Transforms. In *ICALP '07*, volume 4596 of *LNCS*, pages 399–410. Springer-Verlag, 2007. 14, 15, 22, 23, 24

[15] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. In *CCS '93*, pages 62–73, 1993. 7

[16] Mihir Bellare and Phillip Rogaway. Collision-resistant Hashing: Towards Making UOWHF's Practical. In *Cryptgo '97*, volume 1294 of *LNCS*, pages 470–484. Springer-Verlag, 1997. 13, 25

[17] Mihir Bellare and Phillip Rogaway. *Code-Based Game-Playing Proofs and the Security of Triple Encryption*, 2004. (eprint.iacr.org/2004/331). 10

[18] Mihir Bellare and Tadayoshi. Hash Function Balance and its Impact on Birthday Attacks. In *Eurocrypt '04*, volume 3027 of *LNCS*, pages 401–418. Springer-Verlag, 2004. 3

[19] Dan Bernstein. *CubeHash Specification*, 2008. (cubehash.cr.yp.to). 26

[20] Daniel J. Bernstein, Tanja Lange, Christiane Peters, and Peter Schwabe. *Really Fast Syndrome-based Hashing*, 2011. (eprint.iacr.org/2011/074). 27

[21] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sponge Functions. In *ECRYPT Hash Workshop*, 2007. 7, 24

[22] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In *Eurocrypt '08*, volume 4965 of *LNCS*, pages 181–197. Springer-Verlag, 2008. 24

[23] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. *The Keccak Sponge Function Family*, 2008. (keccak.noekeon.org). 24

[24] John Black, M Cochran, and Thomas Shrimpton. On the Impossibility of Highly-Efficient Blockcipher-Based Hash Functions. *Journal of Cryptology*, 22:311–329, 2009. 26

[25] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In *Crypto '02*, volume 2442 of *LNCS*, pages 320–335. Springer-Verlag, 2002. 26

[26] Johannes Buchmann and Sachar Paulus. A One Way Function Based on Ideal Arithmetic in Number Fields. In *Crypto '97*, volume 1294 of *LNCS*, pages 385–394. Springer-Verlag, 1997. 27

[27] Ran Canetti, Oded Goldreich, and Shai Halevi. The Random Oracle Methodology, Revisited. *Journal of the ACM*, 51(4):557–594, 1998. 7

[28] Donghoon Chang. *Preimage Attacks on CellHash, SubHash and Strengthened Versions of CellHash and SubHash*, 2006. (eprint.iacr.org/2006/412.pdf). 28

[29] Denis Charles, Kristin Lauter, and Eyal Goren. Cryptographic Hash Functions from Expander Graphs. *Journal of Cryptology*, 22(1):93–113, 2007. 27

[30] David Chaum, Eugene van Heijst, and Birgit Pfitzmann. Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer. In *Crypto '91*, volume 576 of *LNCS*, pages 470–484. Springer-Verlag, 1991. 27

[31] Scott Contini, Arjen Lenstra, and Ron Steinfeld. VSH, an Efficient and Provable Collision-Resistant Hash Function. In *Eurocrypt '06*, volume 4004 of *LNCS*, pages 165–182. Springer-Verlag, 2006. 27

[32] Scott Contini, Ron Steinfeld, Josef Pieprzyk, and Krystian Matusiewicz. A Critical Look at Cryptographic Hash Function Literature. In *ECRYPT Hash Function Workshop*, 2007. 6, 15

[33] Jean-Sebastien Coron, Yevgeniy Dodis, Cecile Malinaud, and Prshant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In *Crypto '05*, volume 3621 of *LNCS*, pages 430–448. Springer-Verlag, 2005. 8, 9, 11, 12, 20, 22

[34] Jean-Sebastien Coron and Antoine Joux. *Cryptanalysis of a Provably Secure Cryptographic Hash Function*, 2004. (eprint.iacr.org/2004/013.pdf). 27

[35] Joan Daemen, Rene Govaerts, and Joos Vandewalle and. A Framework for the Design of One-Way Hash Functions Including Cryptanalysis of Damgård's One-Way Function Based on a Cellular Automaton. In *Asiacrypt '91*, volume 739 of *LNCS*, pages 82–96. Springer-Verlag, 1991. 28

[36] Joan Daemen, Rene Govaerts, and Joos Vandewalle. A Hardware Design Model for Cryptographic Algorithms. In *ESORICS '92*, volume 648 of *LNCS*, pages 419–434. Springer-Verlag, 1992. 28

[37] Ivan Damgåard. Collision Free Hash Functions and Public Key Signature Schemes. In *Eurocrypt '87*, volume 304 of *LNCS*, pages 203–216. Springer-Verlag, 1987. 5, 6

[38] Ivan Damgård. A Design Principle for Hash Functions. In *Crypto '89*, volume 435 of *LNCS*, pages 416–427. Springer-Verlag, 1989. 6, 15, 16, 25, 27, 28

[39] Ivan Damgård, Lars Knudsen, and Soren Thomsen. DAKOTA – Hashing from a Combination of Modular Arithmetic and Symmetric Cryptograph. In *ACNS '08*, volume 5037 of *LNCS*, pages 144–155. Springer-Verlag, 2008. 27

[40] Quynh Dang. NIST Special Publication 800-106: Randomized Hashing for Digital Signatures. Technical report, National Institute of Standards and Technology, 2009. 22

[41] Richard Drews Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, 1999. 18

[42] Shaojiang Deng, Yantao Li, and Di Xiao. Analysis and Improvement of a Chaos-based Hash Function Construction. *Communications in Nonlinear Science and Numerical Simulation*, 15(5):1338–1347, 2009. 28

[43] Whitfield Diffie and Martin Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. 3

[44] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes. In *Crypto '04*, volume 3152 of *LNCS*, pages 494–510. Springer-Verlag, 2004. 12

[45] Yevgeniy Dodis and Prashant Puniya. Getting the Best Out of Existing Hash Functions; or What if We Are Stuck with SHA? In *ACNS '08*, volume 5037 of *LNCS*, pages 156–173. Springer-Verlag, 2008. 12

[46] Yevgeniy Dodis, Thomas Ristenpart, and Thomas Shrimpton. Salvaging Merkle-Damgård for Practical Applications. In *Eurocrypt '09*, volume 5479 of *LNCS*, pages 371–388. Springer-Verlag, 2009. 11

[47] Orr Dunkelman and Eli Biham. A Framework for Iterative Hash Functions – HAIFA. In *2nd NIST Cryptographich Hash Workshop*, 2006. 22

[48] Orr Dunkelman and Bart Preneel. Generalizing the Herding Attack to Concatenated Hashing Schemes. In *ECRYPT Hash Function Workshop*, 2007. 19

[49] Lei Duo and Chao Li. *Improved Collision and Preimage Resistance Bounds on PGV Schemes*, 2006. (eprint.iacr.org/2006/462). 26

[50] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. *The Skein Hash Function*, 2008. (www.skein-hash.info). 25

[51] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. John Wiley & Sons, 2003. 17

[52] Pierre-Alain Fouque and Gaetan Leurent. Cryptanalysis of a Hash Function Based on Quasi-cyclic Codes. In *CT-RSA '08*, volume 4964 of *LNCS*, pages 19–35. Springer-Verlag, 2008. 27

[53] Praveen Gauravaram, William Millan, Ed Dawson, and Kapali Viswanathan. Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction. In *CISP '08*, volume 4058 of *LNCS*, pages 407–420. Springer-Verlag, 2006. 20

[54] Michael Gorski, Stefan Lucks, and Thomas Peyrin. Slide Attacks on a Class of Hash Functions. In *Asiacrypt '08*, volume 5350 of *LNCS*, pages 143–160. Springer-Verlag, 2008. 24

[55] Shai Halevi and Hgo Krawczyk. The RMX Transform and Digital Signagures. In *2nd NIST Hash Workshop*, 2006. 21

[56] Shai Halevi and Hugo Krawczyk. Strengthening Digital Signatures via Randomized Hashing. In *Crypto '06*, volume 4117 of *LNCS*, pages 41–59. Springer-Verlag, 2006. 14, 21

[57] Shoichi Hirose. How to Construct Double-Block-Length Hash Functions. In *2nd NIST Cryptographic Hash Workshop*, 2006. 26

[58] Shoichi Hirose, Je Hong Park, and Aaram Yun. A Simple Variant of the Merkle-Damgård Scheme with a Permutation. In *Asiacrypt '08*, volume 4833 of *LNCS*, pages 113–129. Springer-Verlag, 2008. 21

[59] Daniel Joscak and Jiri Tuma. Multi-block Collisions in Hash Functions Based on 3C and 3C+ Enhancements of the Merkle-Damgård Construction. In *ICISC '06*, volume 4296 of *LNCS*, pages 257–266. Springer-Verlag, 2006. 20

[60] Antoine Joux. Multicollisions in Iterated Hash Functions: Application to Cascaded Constructions. In *Crypto '04*, volume 31252 of *LNCS*, pages 306–316. Springer-Verlag, 2004. 17

[61] John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In *Eurocrypt '06*, volume 4004 of *LNCS*, pages 183–200. Springer-Verlag, 2006. 7, 18, 19

[62] John Kelsey and Bruce Schneier. Second Preimages on n-bit Hash Functions for Much Less than 2^n work. In *Eurocrypt '05*, volume 3494 of *LNCS*, pages 474–490. Springer-Verlag, 2005. 18

[63] Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search (an analysis of DESX). *Journal of Cryptology*, 14(1):17–35, 2001. 10

[64] Vlastimil Klima. *Tunnels in Hash Functions: MD5 Collisions Within a Minute*, 2006. (eprint.iacr.org/2006/105). 4

[65] Xucjia Lai and James Massey. Hash Functions Based on Block Ciphers. In *Eurocrypt '93*, volume 658 of *LNCS*, pages 55–70. Springer-Verlag, 1993. 16

[66] Jooyoung Lee and Je Hong Park. *Adaptive Preimage Resistance and Permutation-based Hash Functions*, 2009. (eprint.iacr.org/2009/066). 12

[67] Wonil Lee, Donghoon Chang, Sangjin Lee, Soohak Sung, and Mridul Nadi. New Parallel Domain Extenders for UOWHF. In *Asiacrypt '03*, volume 2894 of *LNCS*, pages 208–227. Springer-Verlag, 2003. 25

[68] Stefan Lucks. A Failure-Friendly Design Principle for Hash Functions. In *Asiacrypt '05*, volume 3788 of *LNCS*, pages 474–494. Springer-Verlag, 2005. 19

[69] Stephane Manuel and Nicolas Sendrier. XOR-Hash: A Hash Function Based on XOR. In *WEWRC '07*, 2007. 25

[70] Mahmoud Maqableh, Azman Samsudin, and Mohammed Alia. New Hash Function Based on Chaos Theory (CHA-1). *International Journal of Computer Science and Network Security*, 8(2):20–27, 2008. 28

[71] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In *TCC '04*, volume 2951 of *LNCS*, pages 21–39. Springer-Verlag, 2004. 8

[72] Ueli Maurer and Johan Sjödin. Single-key AIL-MACs from any FIL-MAC. In *ICALP '05*, volume 3580 of *LNCS*, pages 472–484. Springer-Verlag, 2005. 15, 24

[73] Alfred Menezes, Paul Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography*, chapter Hash Functions and Data Integrity, pages 321–384. CRC Press, 1996. 6, 26

[74] Ralph Merkle. One Way Hash Functions and DES. In *Crypto '89*, volume 435 of *LNCS*, pages 428–446. Springer-Verlag, 1989. 6, 15, 16

[75] Miodrag Mihaljevie, Yuliang Zheng, and Hideki Imai. A Cellular Automaton Based Fast One-Way Hash Function Suitable for Hardware Implementation. In *PKC '98*, volume 1431 of *LNCS*, pages 217–233. Springer-Verlag, 1998. 28

[76] Mridul Nandi. Toward Optimal Double-Length Hash Functions. In *Indocrypt '05*, volume 3797 of *LNCS*, pages 77–89. Springer-Verlag, 2005. 26

[77] Moni Naor and Noti Yung. Universal One-Way Hash Functions and their Cryptographic Applications. In *STOC '89*, pages 33–43. ACM, 1989. 13, 25

[78] NIST. *The Keyed-Hash Message Authentication Code (HMAC)*, 2002. (FIPS PUB 198). 15

[79] Yusuke Nito, Kazuki Yoneyama, Lei Wang, and Kazuo Ohta. *How to Prove the Security of Practical Cryptosystems with Merkle-Damgård Hashing by Adopting Indifferentiability*, 2009. (eprint.iacr.org/2009/040). 11

[80] National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register, NIST*, 72(212):62212–62220, 2007. 3

[81] Pinakpani Pal and Palash Sarkar. PARSHA-256 – A New Parallelizable Hash Function and a Multithreaded Implementation. In *FSE '03*, volume 2887 of *LNCS*, pages 347–361. Springer-Verlag, 2003. 25

[82] Jacques Patarin. Collisions and Inversions for Damgård's Whole Hash Function. In *Asiacrypt '95*, volume 917 of *LNCS*, pages 305–321. Springer-Verlag, 1995. 27

[83] Bart Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholiek Universiteit Leuven, 1993. 28

[84] Bart Preneel. Cryptographic Hash Functions. In *3rd Symposium on State and Progress of Research in Cryptography*, pages 161–171, 1993. 27

[85] Bart Preneel. The State of Hash Functions and the NIST SHA-3 Competition. In *Inscrypt '09*, volume 5487 of *LNCS*, pages 1–11. Springer-Verlag, 2009. 28

[86] Bart Preneel, Rene Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In *Crypto '93*, volume 773 of *LNCS*, pages 368–378. Springer-Verlag, 1993. 26

[87] Michael Rabin. *Foundations of Secure Computations*, chapter Digitalized Signatures, pages 155–166. Academic Press, 1978. 16

[88] Mohammed Reyhanitabar, Willy Susilo, and Yi Mu. Analysis of Property-Preservation Capabilities of the ROX and ESh Hash Domain Extenders. In *ACISP '09*, volume 5594 of *LNCS*, pages 153–170. Springer-Verlag, 2009. 14, 23

[89] Mohammed Reza Reyhanitabar, Willy Susilo, and Yi Mu. *Enhanced Security Notions for Dedicated-Key Hash Functions: Defintions and Relationships*, 2010. (eprint.iacr.org/2010/022). 14

[90] Thomas Ristenpart and Thomas Shrimpton. How to Build a Hash Function from any Collision-Resistant Function. In *Asiacrypt '07*, volume 4833 of *LNCS*, pages 147–163. Springer-Verlag, 2007. 11

[91] Ronald Rivest. Abelian Square-Free Dithering for Iterative Hash Functions. In *1st NIST Cryptographic Hash Workshop*, 2005. 22

[92] Ronald Rivest, Benjamin Agre, daniel Bailey, Christopher Crutchfield, Yevgeniy Dodis, Kermin Elliott Fleming, Asif Khan, Jayant Krishnamurthy, Yuncheng Lin, Leo Reyzin, Emily Shen, Jim Sukha, Drew Sutherland, Eran Tromer, and Yiqun Lisa yin. *The MD6 Hash Function: a Proposal to NIST for SHA-3*, 2008. (groups.csail.mit.edu/cis/md6). 25

[93] Phillip Rogaway. Formalizing Human Ignorance: Collision-Resistant Hashing Without the Keys. In *Vietcrypt '06*, volume 4341 of *LNCS*, pages 211–228. Springer-Verlag, 2006. 5, 13

[94] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In *FSE '04*, volume 3017 of *LNCS*, pages 371–388. Springer-Verlag, 2004. 6, 13, 23

[95] Phillip Rogaway and John Steinberger. Constructing Cryptographic Hash Functions from Fixed-Key Blockciphers. In *Crypto '08*, volume 5157 of *LNCS*, pages 433–450. Springer-Verlag, 2008. 26

[96] Markku-Juhani Saarinen. Linearization Attacks Against Syndrome Based Hashes. In *Indocrypt '07*, volume 4859 of *LNCS*, pages 1–9. Springer-Verlag, 2007. 27

[97] Palash Sarkar. Masking Based Domain Extenders for UOWHFs: Bounds and Constructions. In *Asiacrypt '04*, volume 3329 of *LNCS*, pages 187–200. Springer-Verlag, 2004. 25

[98] Palash Sarkar and Paul Scellenberg. A Parallel Algorithm for Extending Cryptographic Hash Functions. In *Indocrypt '01*, volume 2247 of *LNCS*, pages 40–49. Springer-Verlag, 2001. 25

[99] Victor Shoup. A Composition Theorem for Universal One-Way Hash Functions. In *Eurocrypt '00*, volume 1807 of *LNCS*, pages 445–452. Springer-Verlag, 2000. 23

[100] Thomas Shrimpton and Martijn Stam. Building a Collision-Resistant Compression Function from Non-compressing Primitives. In *ICALP '08*, volume 5126 of *LNCS*, pages 643–654. Springer-Verlag, 2008. 26

[101] Martijn Stam. Beyond Uniformity: Better Security/Efficiency Tradeoffs for Compression Functions. In *Crypto '08*, volume 5157 of *LNCS*, pages 397–412. Springer-Verlag, 2008. 26

[102] Martijn Stam. Blockcipher-Based Hashing Revisited. In *FSE '09*, volume 5665 of *LNCS*, pages 69–83. Springer-Verlag, 2009. 26

[103] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, and David Molnar. Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certification. In *Crypto '09*, volume 5677 of *LNCS*, pages 55–69. Springer-Verlag, 2009. 4

[104] D Stinson and J Upadhyay. *On the Complexity of the Herding Attack and Some Related Attacks on Hash Functions*, 2010. (eprint.iacr.org/2010/030). 19

[105] Douglas Stinson. Some Observations on the Theory of Cryptographic Hash Functions. *Designs, Codes and Cryptography*, 38(2):259–277, 2006. 6

[106] Jean-Pierre Tillich and Gilles Zemor. Collisions for the LPS Expander Graph Hash Function. In *Eurocrypt '08*, volume 4965 of *LNCS*, pages 254–269. Springer-Verlag, 2008. 27

[107] Gene Tsudik. Message Authentication with One-Way Hash Functions. *ACM SIGCOMM Computer Communication Review*, 22:29–38, 1992. 17

[108] John von Neumann. *The World of Physics: A Small Library of the Literature of Physics from Antiquity to the Present*, chapter The General and Logical Theory of Automata, pages 606–607. Simon and Schuster, New York, 1987. (Originally presented at the Hixon Symposium on September 20, 1948, at the California Institute of Technology). 28

[109] X. Wang, D. Feng, X. Lai, and H. Yu. *Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD*, 2004. (eprint.iacr.org/2004/199). 3

[110] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In *Crypto '05*, volume 3621 of *LNCS*, pages 17–36. Springer-Verlag, 2005. 3, 15, 17

[111] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In *Eurocrypt '05*, volume 3494 of *LNCS*, pages 19–35. Springer-Verlag, 2005. 3, 15, 17

[112] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002. 28

[113] Kan Yasuda. How to Fill Up Merkle-Damgård Hash Function. In *Asiacrypt '08*, volume 5350 of *LNCS*, pages 272–289. Springer-Verlag, 2008. 6, 14

[114] Kan Yasuda. A Double-Piped Mode of Operation for MACs, PRFs and PROs: Security beyond the Birthday Barrier. In *Eurocryp '09*, volume 5479 of *LNCS*, pages 242–259. Springer-Verlag, 2009. 20

[115] Xun Yi. Hash Function Based on Chaotic Tent Maps. *IEEE Transactions on Express Briefs*, 52(6):354–357, 2005. 28

[116] Kazuki Yoneyama, Satoshi Miyagawa, and Kazuo Ohta. Leaky Random Oracle (Extended Abstract). In *ProvSec '08*, volume 5324 of *LNCS*, pages 226–240. Springer-Verlag, 2008. 11

[117] Gideon Yuval. How to Swindle Rabin. *Cryptologia*, 3(3):187–191, 1979. 3